# OBFUSCATION TECHNIQUES FOR ENHANCING SOFTWARE SECURITY

**Publication number:** JP2002514333 (T)
**Publication date:** 2002-05-14
**Inventor(s):**
**Applicant(s):**
**Classification:**
**- international:** ***G06F1/00; G06F12/00; G06F21/00; G06F21/22; G06F21/24; G09C1/00; G09C5/00;*** **G06F1/00; G06F12/00; G06F21/00; G06F21/22; G09C1/00; G09C5/00;** (IPC1-7): G06F1/00; G06F12/00; G09C5/00
**- European:** G06F21/00N7P1
**Application number:** JP19990508660T 19980609
**Priority number(s):** NZ19970328057 19970609; WO1998US12017 19980609

**Also published as:**
WO9901815 (A1)
US6668325 (B1)
EP0988591 (A1)
CN1260055 (A)
CA2293650 (A1)
AU7957998 (A)

<< less

Abstract not available for JP 2002514333 (T)
Abstract of corresponding document: **WO 9901815 (A1)**

The present invention provides obfuscation techniques for enhancing software security. In one embodiment, a method for obfuscation techniques for enhancing software security includes selecting a subset of code (e.g., compiled source code of an application) to obfuscate, and obfuscating the selected subset of the code. The obfuscating includes applying an obfuscating transformation to the selected subset of the code. The transformed code can be weakly equivalent to the untransformed code. The applied transformation can be selected based on a desired level of security (e.g., resistance to reverse engineering). The applied transformation can include a control transformation that can be creating using opaque constructs, which can be constructed using aliasing and concurrency techniques. Accordingly, the code can be obfuscated for enhanced software security based on a desired level of obfuscation (e.g., based on a desired potency, resilience, and cost).

## CLAIMS

[Claim(s)]

1. it is a method for confusion-izing a code which a computer carries out -- stage; which chooses a subset of a code which should be carried out [ **** ]-izing
A stage which chooses confusion-ized conversion which should be applied; it reaches. A stage which applies conversion;
How to change by ****** and to provide weak equivalence [ as opposed to an unconverted code, in a changed code ].

2. Stage of identifying the singular number or two or more source code input files corresponding to source code for code of application which should be processed;
A stage which chooses a confusion-ized level (effect) needed;
A stage which chooses the maximum execution time or a space penalty (cost);
A stage which reads and analyzes the syntax of an input file;
A stage of providing information which identifies a data type, a data structure, and control structure which are used by application which should be processed;
A stage which chooses and applies confusion-ized conversion to a source-code object until effect needed is chosen or it exceeds the maximum cost; it reaches. A stage which outputs a changed code of application;
A method which is included in a pan and which is enforced by computer according to claim 1.

3. Method according to claim 1 by which ambiguous construct is contained in conversion and this ambiguous construct is built using aliasing and concurrency technology.

4. Method according to claim 1 of including further stage which outputs information about code by which changed application to information about confusion-ized conversion applied to confusion-ized code and source code of the application was confusion-ized.

5. Method according to claim 1 chosen so that conversion may save observability action of code of one application.

6. A code, including a confusion--ization-canceled stage further for confusion-ized release of a code.A method according to claim 1 by which a stage of removing all confusion-ization from a code by which one application was confusion-ized by using a slicing, partial evaluation, data

flow analysis, or statistical analyses is included.

7. -- it is the computer program materialized on a medium in which computer reading for confusion-izing a code is possible -- logic; which chooses a subset of a code which should be carried out [ **** ]-izing
Logic which chooses confusion-ized conversion which should be applied; it reaches. Logic which applies conversion;
A computer program which an implication and a changed code provide with weak equivalence over an unconverted code.

8. Logic which identifies the singular number or two or more source code input files corresponding to source code for code of application which should be processed;
Logic which chooses a confusion-ized level (effect) needed;
Logic which chooses the maximum execution time or a space penalty (cost);
Logic which reads and analyzes the syntax of an input file;
Logic which provides information which identifies a data type, a data structure, and control structure which are used by application which should be processed;
Logic which chooses and applies confusion-ized conversion to a source-code object until effect needed is attained or it exceeds the maximum cost; it reaches. Logic which outputs a changed code of application;
The computer program according to claim 7 included in a pan.

9. Computer program according to claim 7 by which ambiguous construct is contained in conversion and this ambiguous construct is built using aliasing and concurrency technology.

10. The computer program according to claim 7 which includes further logic which outputs information about a code by which changed ABURIKESHON to information about confusion-ized conversion applied to a confusion-ized code and a source code of the application was confusion-ized.

11. The computer program according to claim 7 chosen so that conversion may save an observability action of a code of one application.

12. A code, including confusion--ization-canceled logic further for confusion-ized release of a code. The computer program according to claim 7 in which a stage of removing all confusion-ization from a code by which one application was confusion-ized by using a slicing, partial evaluation, data flow analysis, or statistical analyses is included.

13. -- it is equipment for confusion-izing a code -- means [ for choosing a subset of a code which should be carried out / **** /-izing ];
A means for choosing confusion-ized conversion which should be applied; it reaches. A means for applying conversion;
Equipment with which an implication and a changed code provide weak equivalence over an unconverted code.

14. A means for identifying the singular number or two or more source code input files corresponding to a source code for a code of application which should be processed;
A means for choosing a confusion-ized level (effect) needed;
A means for choosing the maximum execution time or a space penalty (cost);
A means for reading and analyzing the syntax of an input file;
A means for providing information which identifies a data type, a data structure, and control structure which are used by application which should be processed;
A means for choosing and applying confusion-ized conversion to a source-code object until effect needed is attained or it exceeds the maximum cost;
A means for outputting a changed code of application;
The equipment according to claim 13 included in a pan.

15. The equipment according to claim 13 with which an ambiguous construct is contained in conversion and this ambiguous construct is built using aliasing and concurrency technology.

16. The equipment according to claim 13 which contains further a means for outputting information about a code by which changed application to information about confusion-ized conversion applied to a confusion-ized code and a source code of the application was confusion-ized.

17. The equipment according to claim 13 chosen so that conversion may save an observability action of a code of one application.

18. A code, including a means for confusion--ization-canceling further for confusion-ized release of a code. The equipment according to claim 13 with which a stage of removing all confusion-ization from a code by which one application was confusion-ized by using a slicing, partial evaluation, data flow analysis, or statistical analyses is included.

19. The equipment according to claim 13 with which a code contains a Java™ byte code.

20. The equipment according to claim 13 with which conversion provides formation of data confusion, formation of control confusion, or prevention confusion-ization.

---

[Translation done.]

## DETAILED DESCRIPTION

[Detailed Description of the Invention]
Field of confusion-ized technical invention for reinforcing software security This invention relates the interpretation of software, decoding, or reverse engineering to the method and equipment prevention or for blocking at least. Although not being exclusive, speaking more specifically, this invention, It is related with the method and equipment for increasing the structural and logical complexity of software by inserting, removing or rearranging an identifiable structure or information in the form where decompile or a reverse engineering process becomes still more difficult, from software.
The background of invention Software is analyzed by the 3rd person of the character top, and is easy to be copied. In order to reinforce software security until now, great efforts were paid, and these success was various. The problem of such security is related with a desire to hide the programming technique which can be determined via the necessity of preventing the unauthorized copy of software, and reverse engineering.

For example, the established legal method like copyright provides the protection measure on legislation. However, expense and the time of both the things for which the legal right made under such a system is asserted may be these work. The protection given to software under copyright does not cover programming technique. This technology (namely, function contrary to the form of software) is difficult to act as law Mamoru Ueyasu.the software with which a reverse engineer is related from the beginning based on knowledge with a detailed function of the software in question -- ****** -- infringement can be escaped by things. This knowledge can be drawn from analyzing a data structure, abstraction, and the organization of a code.
A software patent provides more extensive protection. However, it is clearly advantageous to combine the legal protection of computer software with technical protection.

The conventional approach against protection of the software in which an ownership opinion is possible was either of the things based on [ using the solution on the hardware based on encryption ] simple rearrangement of source code structure. The technology based on hardware does not have ideal it at the point that it is what generally starts as for expense and is connected to a specific plat form or the Hardware add-on. The solution by software carries out entailment of a trivial code confusion-ized machine called the Crema confusion-ized machine for Java™ standardly. Some confusion-ized machines usually remove source code formatting and a comment for the purpose of the lexical structure of application, and rename a variable. However, such confusion-ized technology does not provide sufficient protection to malicious reverse engineering. That is, reverse engineering is a problem unrelated to the form where software is distributed. When software is distributed in the hardware dependence nature format which there is much information in an original source code, or holds all, a problem gets worse further. As an example of this format, there is a Java™ byte code and architecture neutral distribution format (ANDF).

The programmer may supply great time, efforts, and skill to software development. It is very important that a competitor can be commercially prevented from copying the technology in

which an ownership opinion is possible.

Indication of invention This invention like the method enforced by computer for heightening the resistance force of the software over reverse engineering (or since a useful choice is provided to the general public), The method and equipment for the confusion-ized technology for software security are provided. To the method enforced by computer in one embodiment in order to confusion-ize a code. Entailment of the stage which returns to the stage tested about the completion of supply of confusion-ized conversion to one piece or the code beyond it, the stage which chooses the subset of the code which should be carried out [ **** ]-izing, the stage which chooses the confusion-ized conversion which should be applied, the stage which applies conversion, and a completion testing phase is carried out.

In one modification embodiment, this invention is performed on a computer, In the way the software which is memorized or is operated by that cause controls a computer by the form where the reverse engineering tolerance of the grade which was defined beforehand and controlled is shown, It is a stage which applies the confusion-ized conversion chosen as the portion as which software was chosen, The reverse engineering tolerance of the grade needed, It is related with the method of carrying out entailment of the stage which updates software in order to reflect stage; which attains confusion-ization of a fixed level using confusion-izing chosen so that the validity in the operation of software and the changed size of software might be provided, and confusion-ized conversion.

In the method enforced by computer for this invention to reinforce the security of software in a desirable embodiment, For the application which should be processed. The stage which chooses the stage; maximum execution time or the space penalty (for example, cost) which chooses the confusion-ized level (for example, effect) stage; needed [ of identifying one piece or the source code input file beyond it corresponding to source software ]; Arbitrarily by a source code. The stage which reads and analyzes the syntax of an input file with the arbitrary libraries or additional files which were read directly or indirectly; The data type used by the application which should be processed, The stage of building a suitable table in order to provide the information which identifies a data structure and control structure and to memorize this information, The stage of pretreating the information about application in response to a preprocessing stage, The stage which chooses and applies confusion-ized code conversion to a source-code object; a method including the stage which repeats a confusion-ized code conversion stage, and the stage which outputs changed software is provided until the effect needed is attained or it exceeds the maximum cost.

Preferably, the information about application is acquired using various static analytical skills and dynamic analytical skills. As static analytical skills, the data flow analysis between procedures and data subordinacy analysis are included. As dynamic analytical skills, profiling is contained and information can be arbitrarily acquired via a user. Profiling can be used for determining the confusion-ized level which can be applied to a specific source-code object. The conversion can include the control conversion created using two or more ambiguous constructs (syntax). This ambiguous construct is which a mathematical object cheap, although seen and performed from the standpoint of performance and expensive although a confusion-ized machine can build easily and a confusion release machine moreover fractures it. Preferably, an ambiguous construct is

built using aliasing and concurrency technology, and it deals in it. The information about source application can be acquired using the pragmatic (practical use) analysis which determines similarly the character of the idiom for a program which the application contains, and a language construct.

The effect of confusion-ized conversion can be evaluated using software complexity metric. Confusion-ized code conversion is applicable to all language constructs. a module, a class, or a subroutine can be divided or annexed --; -- new control and data structure can also be created --; -- original control and a data structure are also correctable again. The new construct preferably added to the changed application is chosen so that it may become what was similar in whether it is made to the thing in source application based on the pragmatic information collected between pretreatments. This method can generate a supplementary file including the information about the code by which changed application to the information and source software with which confusion-ized conversion was applied to it was confusion-ized.

Preferably, when P is unconverted software and P' is changed software, confusion-ized conversion is a form where P and P' has the same observability action, and it is chosen so that the observability action of software may be maintained. When being unable to end P or ending with error condition, speaking more specifically, even if P' ends, it is not necessary to carry out it otherwise, and P' ends and generates the same output as P. Although the effect which a user experiences is included as an observability action, P and P' can run with a different detailed action which cannot be observed for a user. For example, as a detailed action of P which it differs and is obtained, and P', file creation, memory use, and network communication are included.

In one embodiment, a slicing, partial evaluation, data flow analysis, or statistical analyses are similarly used by this invention.
Therefore, the tool for confusion release adopted in order to remove confusion-ization from confusion-ized finishing application is provided.

Brief explanation of the drawings Below, with reference to Drawings, this invention is explained as a mere example.
Drawing 1 shows the data processing system according to instruction of this invention.

Drawing 2 illustrates the classification of the protection of software containing the category of confusion-ized conversion.

Drawing 3 a and 3b show the technology for providing software security by the (a) server side execution and the (b) partial server side execution.

Drawing 4 a and 4b show the technology for providing software security by using the native code by which (a) encryption and (b) signature were carried out.

Drawing 5 shows the technology for providing software security through confusion-izing.

Drawing 6 illustrates the architecture of an example of a confusion-ized tool suitable for using it with Java™ application.

Drawing 7 is the table which tabulated the known software complexity metric selection.

Drawing 8 a and 8b have illustrated the elasticity of confusion-ized conversion.

Drawing 9 shows the ambiguous predicate of a different type.

drawing 10 a and 10b -- (a) -- a trivial ambiguous construct -- and (b)
The example of a weak ambiguous construct is provided.

Drawing 11 shows an example of calculation conversion (branch insertion conversion).
a-12 d of drawing 12 has illustrated loop condition insertion conversion.
Drawing 13 illustrates the conversion which changes a reducible flow graph into an irreducible flow graph.

Drawing 14 shows that a code Type can be parallelized, when it does not include data subordinacy at all.

It is shown that drawing 15 can divide the code Type which does not include data subordinacy at all to a parallel thread by inserting a suitable synchronization unmodified instruction.

Drawing 16 shows how Procedures P and Q are made in-line at the call site, and then are removed from a code.

Drawing 17 has illustrated the in-line processing method call.

Drawing 18 shows the technology for carrying out interleave of the two methods declared within the same class.

Drawing 19 shows the technology for creating a different version of the plurality of one method by applying a different confusion-ized conversion set to an original code.

Drawing 20 a - 20c provide the example of the roll conversion including (a) loop blocking, (b) loop unrolling, and (c) loop division.

Drawing 21 shows the example of variable division.

Drawing 22 provides one function built so that a string "AAA", "BAAAA", and "CCB" might be confusion-ized.

Drawing 23 shows the example by which the two 32 bit variables x and y are annexed to one 64 bit-variable z.

Drawing 24 shows the example of the data conversion for array reconstruction.

Drawing 25 illustrates correction of inheritance hierarchy.

Drawing 26 illustrates the ambiguous predicate built from the object and the alias.

Drawing 27 provides an example of the ambiguous construct which used the thread.

Drawing 28 a - drawing 28 d show the original program with which (a) contains three sentence $S_{1\text{-}3}$ under confusion-izing, The confusion release machine with which (b) identifies a "constancy" ambiguous predicate is shown, and the confusion release machine with which (c) determines the common code in a sentence is shown, It is a figure showing the confusion release machine with which (d) applies some final simplification and returns a program to the original form which illustrates the relation of confusion-ized opposite confusion release.

Drawing 29 shows the architecture of a Java™ confusion release tool.

Drawing 30 shows the example of statistical analyses used for evaluation.

Drawing 31 a and 31b provide the table of the outline of various confusion-ized conversion.

Drawing 32 provides the outline of various ambiguous constructs.
Detailed description of the invention The following description is provided in relation to the Java™ confusion-ized tool which an applicant is developing now. However, if it is a person skilled in the art, the technology concerned of the ability to apply also to other programming languages is clear, and it should not be considered that this invention is what is restricted to Java™ application. It considers that the enforcement relevant to the programming language of others of this invention enters in a person's skilled in the art view. The following embodiments expect precision and aim specific at the Java™ confusion-ized tool.

The following names will be used in the following description. that is, P is the input application which should be confusion-ized ;P; whose 'is changed application -- T is conversion in the form where it changes P into P'. P(T) P' is confusion-ized conversion in case P and P' has the same observability action. Generally an observability action is defined as an action which a user experiences. In this way, P' may have creation, then the unexpected effect which was said for the file which P does not have, unless a user experiences it. P and P' does not necessarily need to have equivalent efficiency.

Example drawing 1 of hardware illustrates the data processing system according to instruction of this invention. Drawing 1 shows the computer 100 including three main elements. Entailment of the input-and-output (I/O) circuit 120 used for communicating information with the form appropriately structurized from the portion of others of this computer and this portion is carried out to the computer 100. Entailment of the control processing unit (CPU) 130 in the I/O circuit 120 and the memory 140 (for example, volatility and nonvolatile memory), and a communicating state is carried out to the computer 100. These elements are standardly looked at by most general

purpose computers.

It actually has intention of the computer 100 so that it may become a thing representing the data processing device of an extensive category.

The raster display monitor 160 was shown by the I/O circuit 120 and the communicating state, and it has ordered displaying the pixel which CPU130 generates. Various cathode-ray tubes (CRT) of arbitrary well-known or the display device of other types can be used as the display device 160. The conventional keyboard 150 is also shown by I/0120 and the communicating state. If it is a person skilled in the art, the computers 100 being some larger systems and obtaining will be understood. For example, the computer 100 may be in one network (for example, thing connected to the Local Area Network (LAN)), and communicating state.

So that the computer 100 can carry out entailment of the confusion-ized circuit for reinforcing software security according to instruction of this invention, or it may understand especially, if it is a person skilled in the art again, It is also possible to carry out this invention in the form of the software performed by computer 100 (the metaphor can store this software in the memory 140, and can be performed on CPU130). For example, it is possible to make it confusion-ize with the confusion-ized machine performed on CPU130 in order to provide confusion-ized finishing program P' memorized in the memory 140 according to one embodiment of this invention in the un-getting[ confused ]-ized program P (for example, application) stored in the memory 140.

The schematic diagram 6 of a detailed description shows the architecture of a Java™ confusion-ized machine. When the method of this invention is followed, a Java™ application class file, c. succession tree by which a bus is carried out with arbitrary library files is built with a symbol table, and the type information about all the symbols and the control flow graph about all the methods are provided. Arbitrarily, the user can provide a profiling data file so that it may be generated by the Java™ profiling tool. This information can be used for guiding a confusion-ized machine so that it may guarantee not being confusion-ized by the conversion whose portion of the applications performed frequently is very expensive. The information about application is collected using the standard compiler technology like the data flow analysis between procedures, and data subordinacy analysis. Some they is provided by specialist skills, if there are some which are provided by the user. This information is used in order to choose and apply suitable code conversion.

Suitable conversion is chosen. The dominant standard used when choosing a great portion of suitable conversion is carrying out entailment of the necessary condition that the selected conversion is mixed with the remaining portion and nature of a code. This can cope with it by encouraging conversion with a high felicity value. Another necessary condition should encourage the conversion which produces confusion-ization of a high level by a low execution time penalty. The latter point is attained by choosing the conversion which makes effect and elasticity the maximum and makes cost the minimum.

The priority of confusion-izing is assigned to a source-code object. It will be reflected how it is important for this to confusion-ize the contents of the source-code object. For example, confusion-ized priority becomes high when a specific source-code object includes dramatically a

sensitive high material which can carry out an ownership opinion. About each method, an execution time rank is determined, and this is equal to 1, when spending much time in performing the method rather than which [ other ].

At this time, application is confusion-ized by establishing the mapping and confusion-ized priority and the execution time rank from a suitable in-house-data structure and each source-code object to suitable conversion. Confusion-ized conversion is applied until confusion-ization needed is attained or it exceeds the maximum execution time penalty. Changed application is written in at this time.

The output of a confusion-ized tool is new application equivalent to an original copy functionally. This tool can also generate the Java™ source file to which the information on how the information to which conversion was applied about it, and the confusion-ized code are connected with original application was attached as notes.
Here, it pulls and some examples of confusion-ized conversion are described in relation to a continuation Java™ confusion-ized machine.

Confusion-ized conversion can be evaluated and classified according to the quality. The quality of conversion can be expressed according to the effect, elasticity, and cost. It is related to how the effect of conversion has ambiguous P' in a relation with P. Such metric one of all will become comparatively uncertain from being inevitably influenced by human being's recognition capability. For the purpose concerned, it is enough just to take the effect of conversion into consideration as one measure of the usefulness of the conversion. The elasticity of conversion measures how conversion lasts well to the attack from an automatic confusion release machine.

This is the combination of two factors called a programmer's efforts and efforts of a confusion release machine. Elasticity is trivial or can be measured on the graduation of until one-way. One-way conversion is going too far at the point that it cannot be reversed. The 3rd component is conversion execution cost. This is the execution time or the space penalty worn as a result of having used changed application P'. The further details of conversion evaluation are stated in the portion of detailed explanation of following desirable embodiments. The main classifications of confusion-ized conversion are shown in drawing 2 C, and are given for details to e-2 g of drawing 2.

Categorization of the :confusion-ized conversion whose example of confusion-ized conversion is as follows is carried out to the formation of control confusion, the formation of data confusion, the formation of layout confusion, and prevention confusion-ization, and it deals in it. Some of these examples are stated below.
Set conversion, ordering conversion, and calculation conversion are included in control confusion-ization.

The flows of control of a fruit are hidden behind an un-functional sentence unsuitable to calculation conversion, Removing abstraction of the flows of control of introducing [ a corresponding high level language construct ]-by object code level which does not exist at all-code sequence;, and a fruit, or introducing a spurious thing is included.
when the 1st classification (flows of control) is taken into consideration, there is correlation with

the Cyclomer tick and nesting complexity metric strong between the complexity the code of a piece has been perceived to be, and the number of predicates which it contains -- thing suggestion is carried out. An ambiguous predicate enables construction of the conversion which introduces a new predicate in a program.

When drawing 11 a is referred to, ambiguous predicate $P^T$ is $S=S_1$. -- It is inserted into basic block S which is $S_n$. In this way, S is divided into a half. Since a $P^T$ predicate will always be evaluated to "truth", it is an unsuitable code. In drawing 11 b, S is again divided into two halves and these halves are changed into different two confusion-ized finishing version $S^a$ and $S^b$. It becomes therefore, less clear to a reverse engineer that $S^a$ and $S^b$ achieve the same function.

Although drawing 11 c is similar with drawing 11 b, the bug is introduced in $S^b$. A $P^T$ predicate always chooses the right version of code $S^a$.

Confusion-ized conversion of another type is data conversion. An example of data conversion is carrying out the inverted structure interpretation of the array in order to increase the complexity of a code. One array can be divided into two or more sub arrays, and two or more arrays can be annexed to a single array, or the dimension of an array can also be increased or (flattening) decreased again (folding). Drawing 24 shows a fixed number of examples of array conversion. In the sentence (1-2), the array A is divided in the two sub arrays A1 and A2. A2 includes an element with odd indices including the element in which A1 has even indices. The sentence (3-4) has illustrated how interleave is carried out and the two integer arrays B and C get so that they may produce one array BC. The element from B and C is diffused uniformly [ a rear spring supporter ] in the whole changed array. The sentence (6-7) has illustrated folding to the array D1 of the array D. This conversion introduces the data structure which was lacking conventionally, or removes the existing data structure. In this way, for example, when declaring a two dimensional array, since a programmer performs it in the state of mapping on the data in which a selected structure usually corresponds, for one purpose, the ambiguity of a program may increase substantially.

If fold of the array is carried out to 1-d structure, a reverse engineer will be deprived of precious pragmatic information.
Another example of confusion-ized conversion is prevention conversion. in contrast with control or data conversion, the main last purpose of prevention conversion is not to program vaguely for human being's reader, and there is in making known automatic confusion release technology more difficult, or carrying out exploitation of the known problem within the present confusion release machine or decompiler. Such conversion is peculiar respectively and is known as conversion of a target. An example of peculiar prevention conversion is carrying out reorder of for-loop to run backward. Such re-ordering is possible when a loop does not have loop support model data subordinacy at all. The confusion release can conduct the same analysis and can carry out reorder of the loop to forward direction execution. However, when false data subordinacy is added to the reversed loop, discernment of a loop and its re-ordering will be prevented.

The further specific example of confusion-ized conversion is stated in the portion of detailed explanation of following desirable embodiments.

Detailed explanation of a desirable embodiment It is becoming a general **** target to distribute software with the form holding most or all of information that exists in an original source code.

One important example is a Java byte code. Since deconstructivism BAIRU is easy for this code, the danger of an attack of malicious reverse engineering is increased.

Therefore, according to one embodiment of this invention, two or more technology for technical protection of software security is provided. In detailed explanation of a desirable embodiment, about automatic code confusion-ization being the method most implementable now for preventing reverse engineering, we prove and are going to go. Next, we describe the design of the code confusion-ized machine which is a confusion-ized tool which changes a program into an understanding and the equivalent in which it is still more difficult to carry out reverse engineering.

The confusion-ized machine is based on application of code conversion similar to what a KOMBAIRA optimization program uses in many cases. Describe much starting conversion, classify them and The effect (for example, to what extent is human being's reader perplexed?),

They are evaluated about elasticity (for example, can it be equal to the attack of automatic confusion release however?), and cost (for example, is how much performance overhead added to the application?).

The measure considered that it can use a confusion-ized machine at the end to various confusion release technology (for example, program splicing) and them is described.

1. Introduction If sufficient time, efforts, and a decision are given, the able programmer will always be able to do reverse engineering to any applications. The reverse engineer who gained physical access to application can do deconstructivism BAIRU of it (using a disassembler or deconstructivism BAIRA), and then can analyze the data structure and flows of control. Also manually this is made and can also be performed using the reverse engineering tool like a program slicer.

Reverse engineering is not a new problem. However, until reverse engineering is difficult and recent years come (although it is by no means impossible), since it was a native code to which a great portion of program is large-sized, and is monolithic, and is shipped in the state of a strip, A software developer has a problem referred to as not having paid attention so much in reverse engineering.

However, this situation is changing as it becomes a general **** target that carrying out decompile and reverse engineering distributes software with an easy form. As an important example, there is a Java byte code and architecture neutral distribution format (ANDF).

Especially Java application has raised the problem for a software developer. These are distributed on the Internet as a Java class file which is a virtual-machine code of the hardware independence which holds the original Java source information altogether as a matter of fact. Therefore, these class files are easy to decompile. and since many of calculations are performed

in a standard library, a Java program is boiled occasionally, is carried out, and its size is small, therefore it is comparatively apt to carry out reverse engineering.

A Java developer's main concerns are not the thorough re-engineering of all applications. Such action is the Copyright Act clearly. From the ability for [29] to be broken and fight for a lawsuit, it is that it is comparatively valueless. It is expected to say that a competition partner can extract the algorithm and data structure in which an ownership opinion is possible from the application, and that the developer is most afraid can incorporate it now in the program of its company rather.

This not only gives a competition partner commercial validity (it is because development time and cost are reduced), but is also that detection and legal pursuit are difficult. A powerful company with an infinite budget concerning [ the point of this last ] law It is especially applied to the small-scale developer who probably does not have the economic strength which performs the legal battle over a long period of time to [22].

Drawing 2 is provided with the outline of protection of various forms for providing the legal protection of computer software or security. drawing 2 -- (a) -- the classification of target information, (d) layout confusion, (e) data confusion, (f) control confusion, and (g) prevention confusion is provided by the kind of protection to malicious reverse engineering, the quality of (b) confusion-ized conversion, and (c) confusion-ized conversion.

A software developer argues below about various forms of technical protection of an available intellectual property. Although this argument is restricted to the Java program distributed on the Internet as a Java class file, most results are applied also to other languages and an architecture neutral format so that clearly [ a person skilled in the art ]. The only rational approach against protection of a move code proves a certain thing by code confusion-ization. Furthermore, we present some confusion-ized conversion, classify these according to validity and efficiency, and show how it enables it to use them within an automatic confusion-ized tool.

The remaining portion of detailed explanation of a desirable embodiment is constituted as follows. The second article shows the outline of technical protection of various forms over the theft of software, and code confusion-ization proves providing the prevention most economical now. Section 3 shows the easy outline of a design of Kava which is a code confusion-ized machine for Java which is under construction now. Section 4 and Section 5 have described the standard used for classifying and evaluating confusion-ized conversion of a different type. Section 6, Section 7, Section 8, and Section 9 have presented the catalog of confusion-ized conversion. Section 10 shows the more detailed confusion-ized algorithm. It has rounded off with Section 11 by the argument about the conclusion of a result, and the future direction of the formation of code confusion.

2. Protection of intellectual property The following plots are considered. Alice is a small-scale software developer who wants the user to enable it to use her application by ****** probably on the Internet. A bob is a developer of the rival who thinks that a commercial predominance can be acquired to Alice, when he is able to access the key algorithm and data structure of application of Alice.

The reverse engineer who considers as work changing this into two adversaries (Alice), i.e., the software developer who is going to protect his code from an attack, and the form which analyze application, reads it easily and can understand it (bob)

Two player games between ** can be considered. here -- a bob -- application -- the original source of Alice -- a little -- or -- please care about the point that changing into a near thing is unnecessary. It is only required for the code by which reverse engineering was carried out to be able to understand for a bob and his programmer. Similarly, please also care about that Alice may not have even the necessity of protecting one's whole application from a bob. Probably this comprises "the bread code with butter" whose most is not actually the center of concern for a competition partner.

Alice can protect its code from the attack of a bob using either of the legal or technical protection as shown in above-mentioned drawing 2 a. It is a difficult thing from economical reality that a small company like Alice makes the competition partner who is more greatly more powerful observe the law although the Copyright Act covers the software structure to be sure. more attractive solution has Alice in protecting one's code, when making it it become almost impossible to make reverse engineering technical very difficult, and to make reverse engineering impossible, or to realize economically at least. The trial of some first stages in technical protection is indicated by Gosler. (James R.Gos1er,Software protection:Myth or reality? In CRYPTO'85 -- Advances in Cryptology, pages August, 157-1985 [ 140-].

The safest approach is Alice's not selling its application at all, but selling the service rather. If it puts in another way, a user will connect with the site in Alice, in order to run a program distantly as the application itself never cannot be accessed, the electronic money of a small sum is paid rather each time and it is shown in drawing 3 a. That a bob gains physical access to the application never does not have an advantage for Alice, therefore it is to be unable to carry out the reverse engineering of it. The minus aspects have application in that performance far worse than the case where it performs locally on a user's site may be shown for the limit concerning network band width and waiting time with a natural thing. Partial solution is dividing application into the private portion run by two portions, i.e., the public part locally run on a user's site as shown in drawing 3 b, and remoteness (the algorithm which wants to protect Alice is included).

Another approach will be that Alice enciphers the code, before sending its code to a user as shown for example, in drawing 4 a. Regrettably, this functions, only when the whole decipherment / execution process are performed within hardware. Such a system, Herzberg (Amir Herzberg.) and Shlomit S.Pinter.Public. protection of software.ACM. Transactions on Computer. Systems, 5. (4): November, 393-1987 [ 371-]. And it is described by Wilhelm (Uwe G.W11he1m.Cryptographically protected objects.http://lsewww.epfl.ch/-wilhelm/CryPO.html.1997). When a code is executed by the virtual-machine interpreter within software (that is most frequently right like [ Java bytecodes ]), it becomes always possible for a bob to monitor and decompile a decoded code.

The Java™ programming language won popularity mainly for the architecture neutral byte code. Although this makes a move code easy clearly, it reduces performance by 1 figure compared with a native code. This drew development of the Just InTime Compiler translated into a native

code while executing a Java byte code as it has expected. Such a translating program was able to be used for it so that Alice may create the native code version of its application about all the general architecture. When downloading application, a user's site must identify the combination of the architecture/operating system which it is running, for example, a corresponding version will be transmitted to drawing 4 b as shown. Only by the ability to access a native code, although the task of a bob is not impossible, it becomes still more difficult. Complexity when transmitting a native code increases further. Unlike the Java byte code from which a problem receives a byte code check before execution, I hear that a native code cannot run safely thoroughly, and there is on a user's machine. When Alice is the member by whom the community was trusted, it can accept her guarantee of not carrying out that no applications of a user are harmful at the user side. Probably, Alice must carry out the digital signature of it to the code under transmission so that it may prove to a user that a code is an original code which oneself wrote in, in order for **** to also confirm that it is not going to pollute application.

There is approach of the last which we take into consideration by code confusion-ization as shown, for example in drawing 5. A fundamental view is a thing of Alice letting the confusion-ized machine which is changed into the application which is what is harder to understand application far for the bob of the functionally same thing as an original copy and which is a program pass, and making it run one's application. We believe that confusion-ization is the technology in which the realization for protection of the software dealings secret that the attention which deserves it should be received from now on is possible.

Unlike the server side execution, protecting one application from malicious reverse engineering efforts thoroughly can never perform code confusion-ization. Since the bob which was able to give sufficient time and decision searches the important algorithm and data structure, it can always examine the application of Alice. A bob may make it run a confusion-ized finishing code through the automatic confusion-ized release program which tries to cancel confusion-ized conversion in order to help these efforts.

Therefore, the security level from the reverse engineering which a confusion-ized machine adds to application, for example, elaborate-izing of the conversion used with (a) confusion-ized machine and (b) -- it is influenced by the quantity of the resources (time and space) which can use the power of an available confusion-ized release algorithm, and (c) confusion release machine. I think that he would like to copy ideally the situation in the present public key cryptosystem with which the dramatic difference of the cost of encryption (it is easy to discover a large prime number) and a decipherment (it is difficult to factor-ize a large number) exists.

Although it is applicable in a polynominal time (polynomial time) so that it may argue below, the confusion-ized conversion which needs an exponential time (exponential time) for confusion--ization-canceling will actually exist there.

3. The engineering drawing 6 of a Java confusion-ized machine shows the architecture of Kava which is a Java confusion-ized machine. The main inputs to a tool are confusion-ized levels which a Java class file set and a user demand. Arbitrarily, a user can provide the file of profiling data as he is generated by the Java profiling tool. Although a confusion-ized machine is guided, this information is usable so that it may check not being confusion-ized by the conversion whose

portion frequently performed among applications is very expensive. The input to a tool is Java application given as one set of a Java class file. A user chooses the maximum execution time / space penalty (cost) allowed for the confusion-ized level (for example, effect) and confusion-ized machine which are needed to add to application similarly. Kava reads and carries out purging of the class file with all the library files by which reference directions were carried out directly or indirectly. A succession tree is thoroughly built with the control flow graph about the symbol table showing the type information about all the symbols, and all the methods.

Kava includes the big code conversion pool described below. However, before these become applicable, the bus for pretreatment collects the information various type about application according to one embodiment. The information on some kinds has a thing in which it can collect using standard compiler technology, such as data flow analysis between procedures, and data subordinacy analysis, and is provided by the user, and deals, and a thing collected using the specialized technology. For example, pragmatic analysis analyzes application in order to find what kind of a language construct and programming idiom contain it.

The information collected between the paths for pretreatment is used in order to choose and apply suitable code conversion. All the types in application of language construct may be an object of confusion-izing. For example, it is possible to correct for a class to be divided or annexed, and to be able to change or create a method, and to create new control and data structure and an original thing. The new construct added to application can be chosen so that it may become what was similar in whether it is made to the thing in source application based on the pragmatic information collected during the prepass.

A translation process is repeated until the effect needed was attained or it exceeds the maximum cost. The output of a tool is new application usually functionally given as a Java class file set equivalent to an original thing. The tool could also generate the Java source file to which the information to which conversion was applied about it, and the information how the confusion-ized finishing code was related to an original code were similarly attached as notes. The source with notes becomes useful because of debugging.

4. Classification of confusion-ized conversion Various confusion-ized conversion is described, and the remaining portion of detailed explanation of this desirable embodiment classifies and estimates it. The concept of confusion-ized conversion is first begun from formation-izing.
Definition 1 (confusion-ized conversion)
P-$^{T}$->P' is considered as legal confusion-ized conversion and the following conditions must be maintained in here.

- When P cannot be ended or it ends by error condition, even if P' ends, it is not necessary to carry out it.

- When that is not right, P' must end and must generate the same output as P.

An observability action is vaguely defined as "an action of ****** which a user passes through and tries." Unless this, i.e., a user, experiences the side effects, P' means that it can have the side effects (for example, creation of a file or message sending on the Internet) which P does not

have. We need to care about the point of not demanding for P and P' to be similarly efficient. As a result of the thing of many of our conversion, P' becomes slower than P or will actually use many memories rather than P.

The main division lines between the classes from which confusion-ized technology differs are shown in drawing 2 c. It classifies [ 1st ] confusion-ized conversion according to the kind of target information first. Some simple conversion is aimed at the lexical structure (layout) of application called source code formatting, and the name of a variable. The more elaborate conversion which is an object of interest is aimed at either the data structure used by application, or its flows of control in one embodiment.

It classifies [ 2nd ] conversion according to the kind of OBERESHON performed about target information. Two or more conversion which operates contrast or a set of data exists so that it may understand, if d-2 g of drawing 2 is seen. This conversion builds abstraction of new falsehood by doubling the data or control which decomposes the abstraction standardly created by the programmer or is unrelated, and making it a bunch.

Similarly, some conversion affects ordering of data or control. The turn that two items are declared in many cases, or two calculations are performed does not have any effects to the observability action of a program. However, the far useful information embedded in the selected order for the programmer who wrote in the program, and the reverse engineer may exist. As two items or a phenomenon is near spatially or in time, the probability that they have a relation in one of forms becomes higher. Ordering conversion tends to investigate this by randomizing an order of declaration or calculation.

5. Before trying the design of confusion-ized conversion of either evaluation of confusion-ized conversion, the quality of this conversion must be able to be evaluated. or [ that it is what adds two or more standards, i.e. how much / them / ambiguity, to a program in this section, or (for example, effect) they cannot destroy easily how for a confusion-ized release machine ] (for example, elasticity) -- and, It tries to classify conversion in accordance with the standard whether they add how much calculation overhead (for example, cost) to confusion-ized finishing application.

5.1 Measure of effect It is defined what it means first for program P' that it is ambiguous (or complicated or reading is impossible) further from the program P.
Such metric one of all the may be comparatively uncertain from it being based on human being's recognition capability (part) on a definition.
Fortunately, many researches in software complexity metric branching of software engineering can be used. In this field, metric one means helping construction of the software which reading is possible and reliability can maintain highly, and it is designed. it is based on boiling metric one occasionally, carrying out it, it calculating various theque CHUA characteristics of a source code, and combining these enumerated data to the measure of complexity. Other things were purely speculative although derived from the experimental research of the formal program with actual some proposed so far.

The formula of the detailed complexity seen in metric document can be used for deriving the general sentence like "except for the point that P' contains more characteristics q compared with P, when the same, P' has the program P and P' still more complicated than P." When such a sentence is given, we can try to build the conversion which adds more q-characteristics to a program, getting to know that a possibility that this will increase the ambiguity is high.
E(X) of drawing 7 is the complexity of the software configuration element X.
F is a function or a method, C is a class, and P is the table which is a program and which tabulated some of more reputable complexity measures.

When used in a software construction project, the standard last purpose is to make these measures into the minimum.
Contrary to this, when confusion-izing a program, we generally think that he would like to make a measure into the maximum.

We can formalize the concept of effect by complexity METORIKKUSU, and this will be used as a measure of the usefulness of conversion below.Privately, one conversion is effective when carrying out the outstanding work which confuses a bob when it hides the intention of the original code of Alice. If it puts in another way, the effect of conversion will measure whether compared with an original code, it is hard for human being to understand a confusion-ized finishing code however. This is formalized in the following definitions. : The definition 2 (conversion effect)T is considered as action preservation conversion, and $P-^{T}->P'$ changes a source program into target program P'. Let E (P) be the complexity of P as one of metric one of the of drawing 7 defines.

The effect of $T_{pot}$ (P), i.e., T to the program P, is the measure of a grade that T changes the complexity of P. It defines as this and $T_{pot}(P) \stackrel{def}{=} E(P')/E(P)-1$. the case where T is $T_{pot}(P) > 0$ -- the effect -- it is powerful confusion-ized conversion.
In this argument, effect is measured on a three-point graduation (low, inside, quantity).
The observations in the table 1 enable us to list some desirable characteristics of the conversion T. T -- the effect -- in order to be powerful confusion-ized conversion, it is required for it to perform the following things.
- Increase the whole program size ($u_1$) and introduce a new class and method ($u^{a}_{7}$).
- Introduce a new predicate ($u_2$) and increase the nesting level ($u_3$) of conditional and a looping construct.
- Increase the number of method arguments ($u_5$), and the instance variable subordinacy between classes ($u^{d}_{7}$).
- Increase the height ($u^{b.c}_{7}$) of a succession tree.
- Increase long range variable subordinacy ($u_4$).

5.2 Measure of elasticity It glances and it seems that it is trivial to increase $T_{pot}$ (P). For example, in order to increase $u_2$ metric, what we should make is only adding some arbitrary if sentences to

```
main() {                    main()  {
S1;                         S1;
S2;     = T = >             if (5==2) S1;
                                  S2;}
                            if (1>2) S2;
P.:                          }
```

Regrettably, such conversion is not helpful as a matter of fact from it being easily canceled by simple automatic technology and getting by it. Therefore, it is required to introduce the concept of the elasticity which measures whether conversion can bear however under the attack of an automatic confusion release machine. For example, the elasticity of the conversion T, : which can be seen as combination of the following two measures -- namely, -- programmer's efforts:, although the automatic confusion-ized release machine which can reduce effectively the effect which is T is built. Quantity of the time needed: Reach. Efforts of a confusion release machine:

Time and space which are needed for this automatic confusion release reducing the effect of T effectively.

It is important to distinguish elasticity and effect. It is elastic when it is effective when it tends to confuse human being's reader, but conversion confuses an automatic confusion-ized release machine.

Elasticity is trivial or is measured on the graduation of until one-way as shown in drawing 8 a. One-way conversion is special in the meaning that they must never have been canceled. This is because the information from the program which is not required in order to execute a program correctly, although these conversion was useful for human being's programmer is removed standardly. As an example, formatting is removed and the conversion which carries out the scramble of the variable identifier is included.
Standardly, although other conversion does not change the observability action, it adds to a program information which increases the "information load" to human being's reader and which is not helpful. These conversion can be canceled with the degree of difficulty to change.

Drawing 8 b shows that efforts of a confusion-ized release machine are classified as either a polynomial time or an IKUSUBONENSHARU time. The work needed for automating a programmer's efforts, i.e., confusion-ized release of the conversion T, is measured as a function of the range of T. This is based on the intuitive ability that it is easier to build the measure to the confusion-ized conversion to which a twist also has only on the small portion of the procedures, to what can have on the whole program.
:, i.e., T, defined using the term which borrowed the range of conversion from the code optimization theory is local transformation when it affects the single basic block of control flow graph (CFG).

When affecting the whole CFG, it is global-area conversion, when affecting the information flow between procedures, it is conversion between procedures, and it is interprocess conversion when affecting the interaction between the control threads which it performs independently.

The definition 3 (conversion elasticity)T is considered as action preservation conversion, and P=T=>P' changes the source program P into target program P'. $T_{res}$ (P) is the elasticity of T to the program P. $T_{res}$ (P) is one-way, when information is removed from P so that P may not be reconstructed from P'. Otherwise, it is $T_{res}^{def}$= elasticity ($T_{DEobfuscator\ effort}$, $T_{programmer\ effort}$) and elasticity is the function defined in the matrix in drawing 8 b here.
5.3 In graphic scale 2b of execution cost, it turns out that effect and elasticity are two of three components which describe one conversion. The 3rd component, i.e., the cost of conversion, is the execution time or the space penalty which conversion invites to confusion-ized finishing application.
we classify cost on a four-point graduation (no charge, cheapness, and a high price -- extraordinary), and define each marks of these below.

Consider the definition 5 (conversion cost)T as action preservation change, and $T_{cost}$(P) **{unreasonableness, becoming a high price, cheapness, and no charge} -- execution of P' -- P -- 0 (1) individuals -- the case where many resources are needed -- $T_{cost}$(P) = no charge ;P execution of '-- P -- 0 (n) individuals -- the case where many resources are needed -- $T_{cost}$(P) = -- inexpensive ;P >1 -- execution of P' -- P -- 0 ($n^p$)
an individual -- the case where many resources are needed -- $T_{cost}$(P) = expensive; -- again -- otherwise, $T_{cost}$(P) = -- being extraordinary (that is, execution of P' needs large resources more nearly exponentially than P) -- it carries out.

The actual cost which accompanies exchange should care about being influenced by the application environment. For example, the simple quota sentence a= 5 inserted in the best level of a program is covered only with a constancy overhead. The same sentence inserted in the inside of an internal loop will have substantial still higher cost. Unless other indication can be found, we are always provided with the cost of conversion as if it was applied to us by the nesting level of the outermost part of a source program.

5.4 Nature measure Here, the formal definition of the quality of confusion-ized conversion can be shown. : Definition 6 (quality of conversion)
:$T_{qual}$(P) = by which nature $T_{qual[\ of\ the\ conversion\ T\ ]}$ (P) is defined as the effect of T, elasticity, and combination of cost ($T_{pot}$ (P), $T_{res}$ (P), $T_{cost}$ (P)).
5.5 Layout conversion Before investigating new conversion, it sees simply about typical trivial layout conversion, for example in present Java confusion-ized machine called Crema. (Hans Peter Van Vliet.Crema....Java confusion-ized machine.) http.//web.inter.nl.net/users/H.P.van.V1 iet/crema, html, January, 1996. The 1st conversion removes sometimes available source code formatting information within a Java class file. This from it being unrecoverable once original formatting passes away. Are one-way conversion and only very slight semantic contents exist in a certain; formatting, Since big confusion is not drawn at all when the information is removed, this is conversion of no charge [ affect / by the last which is conversion of low effect / this / the space and the time complexity of application ].
Scrambling of an identifier name is one-way and free conversion similarly. However, since the

identifier includes many pragmatic information, it has an effect far higher than formatting removal.

6. Control conversion The catalog of confusion-ized conversion is introduced in this section and the following several Section. the some are derived from conversion of the well-known used in other fields, such as compiler optimization and software re-engineering, and other things are developed only for the purpose of confusion-izing according to one embodiment of this invention.

In this section, the flows of control of source application are stated about the conversion which it is going to make ambiguous. We classify these conversion as what affects a set of a control flow, ordering, or calculation as shown in drawing 2 f. Whether mutual imputation being divided logically and the calculation which both does not belong are annexed by control set conversion.

Control ordering conversion randomizes an order that calculation is carried out. The calculation conversion can insert a new (redundant or dead) code, or can make an algorithm change to source application.

Probably, about the conversion which changes flows of control, a constant rate of calculation overheads will be inescapable. For Alice, this means that she may be pressed for selection between a dramatically efficient program and the program confusion-ized extremely. She can be assisted when a confusion-ized machine enables it to perform selection between inexpensive conversion and expensive conversion in this trade-off.

6.1 Ambiguous predicate When designing control change conversion, there is actual SUBJECT also in making them into a tolerance **** thing not only to making it inexpensive but to the attack from a confusion-ized release machine. In order to attain this, it depends for much conversion on an ambiguous variable and existence of an ambiguous predicate. Privately, the variable V is ambiguous when it has some characteristics q which the confusion-ized release machine of what is transcendentally known by the confusion-ized machine cannot deduce easily.

When similarly it is very difficult for a confusion-ized machine for a confusion-ized release machine to deduce the result of what is well-known, the predicate P (Boolean expression) is ambiguous.

It is a key to the control conversion which is main challenges and is extremely elastic for the maker of a confusion-ized tool for a confusion-ized release machine that a break through can create a difficult ambiguous variable and predicate. we measure the elasticity (namely, the tolerance over a confusion-ized release attack) of an ambiguous variable or a predicate on the same graduation as conversion elasticity (namely, trivial one, weakness, strength, and full -- one-way).
we measure the added cost of an ambiguous construct similarly on the same graduation (namely, no charge, cheapness, and a high price -- extraordinary) as conversion cost.

In the point p in a program, the definition 7(ambiguous construct)1 ** variable V is ambiguous, when it has the characteristic q in the point p known for the confusion-ized time. From the

context, in us, p writes this to be $V_q^p$ or $V_q$, when clear. In p, the predicate P is ambiguous, when the result is known for the confusion-ized time. We write it as $P^F_p$ ($P^T_p$), when P always evaluates imitatively (truth) in p, and P sometimes writes them to be $P^?_p$, when sometimes evaluating imitatively, truth and. From the context, p will be omitted also here again, when clear. Drawing 9 shows the ambiguous predicate of a different type. A solid line expresses the course which may sometimes be taken and a dashed line shows the course never taken.

Below, some examples of a simple ambiguous construct are shown. It is easy to build these for a confusion-ized machine, and easy to decode them similarly for a confusion-ized release machine. Section 8 provides the example with far high elasticity of an ambiguous construct.

6.1.1 Trivial and weak ambiguous construct The ambiguous construct is trivial when a confusion-ized release machine can decode it by statistical local analysis (that is, the value can be deduced). Analysis is local when it is restricted to the single basic block of control flow graph. Drawing 10 a and 10b provide the example of a (a) trivial ambiguous construct and a (b) weak ambiguous construct.

Similarly, we consider that this variable is trivial, when one ambiguous variable is calculated to a library function using the simple semantics understood well from a call. About a language like Java™ which is a language which needs all the enforcement for supporting a standard library class set, the starting ambiguous variable is built easily. int$V^S$ whose simple example is a library function with which random (a, b) returns one integer of a...b within the limits [1, 5] - Random (1, 5)

It comes out. Regrettably, confusion-ized release is easy for the starting ambiguous variable in a similar manner. It is only required to carry out pattern matching about a function call within the code which the confusion-ized release machine designer tabulated the semantics of all the simple library functions, and then was confusion-ized.

An ambiguous construct is weakness when a confusion-ized release machine can decode it by static global-area analysis. Analysis is global when it is restricted to single control flow graph. 6.2 calculation conversion : by which calculation conversion goes into the following three categories -- that is, ; which hides the flows of control of a fruit behind the unrelated sentence which does not contribute to actual calculation -- by the object code level in which the corresponding language construct of a high level does not exist at all, a code sequence is introduced, flows-of-control abstraction of a fruit is removed, or a spurious thing is introduced.

6.2.1 Insertion $U_2$ of a dead code or an unrelated code and $U_3$ metric have suggested that correlation strong between the complexity the code of a piece has been perceived to be, and the number of predicates which it contains exists. If an ambiguous predicate is used, the conversion which introduces a new predicate in a program can be devised.

Basic block $S=S_1$ in drawing 11 .... $S_n$ is taken into consideration. In drawing 11 a, slurring-speech $P^T$ is inserted into S and it is fundamentally divided into a half. In order that it may always evaluate a $P^T$ predicate truly, it is an unrelated code. In drawing 11 b, it progresses in order to divide S into two halves also here again, next to create different two confusion-ized

finishing version $S^a$ of this 2nd half, and $S^b$. $S^a$ and $S^b$ will be created by applying a different OBS conversion set to the 2nd half of S. Therefore, it is not directly clear to a reverse engineer that $S^a$ and $S^b$ achieve the actually same function. We use predicate $P^7$, in order to perform selection between $S^a$ and $S^b$ in run time.

Although drawing 11 c is similar with drawing 11 b, it introduces a bug in $S^b$ shortly.
A $P^T$ predicate always chooses right version $S^a$ of a code.

6.2.2 Extended drawing 12 of loop conditions shows what-izing of the loop can be carried out [ **** ] by making conclusion conditions more complicated. In the $P^T$ or $P^F$ predicate which does not have influence, I hear that a fundamental view extends loop conditions to the number of times a loop is due to perform, and it is in it.
For example, the predicate which we added by drawing 12 d is always evaluated truly from it being $X^2(X+1)^2=0 \pmod 4$.

6.2.3 conversion to an irreducible flow graph from reducible It is alike occasionally, it carries out and a programming language is compiled by the native or virtual-machine code which has power of expression further from the language itself. When this is applied, we can devise language division conversion in this way. That conversion is language division conversion is a case where it introduces the virtual-machine (or native code) instruction sequence in which which source language construct does not have direct correspondence nature at all. or [ trying for a confusion-ized release machine to compound an equivalent (however, superimposed) source language program, when faced with this instruction sequence ] -- it must stop or having to abandon all

For example, although a Java™ byte code has a GOTO command, a Java™ language does not have a corresponding GOTO statement at all. It means that arbitrary, this, i.e., a Java™ byte code, flows of control can be expressed, and the Java™ language can, on the other hand, express only the only structurized flows of control (easily). Although the control flow graph generated from the Java™ program always serves as reducible standardly, it can be said that the Java™ byte code can express an irreducible flow graph.

Since it becomes very difficult to treat expression of an irreducible flow graph in a language without GOTO, we build the conversion which converts a reducible flow graph into an irreducible flow graph. This can be performed by changing the structurized loop to a loop with a multiplex header. For example, it is made for it to turn out in drawing 13 a that ambiguous predicate $P^F$ is added to a While loop, and the jump to the center of the loop exists. In fact, this branching is never taken.

Java™ decompiler must stop having to change an irreducible flow graph to what will not reproduce a code or contains a foreign Boolean variable. Alternatively, a confusion-ized release machine guesses that all the irreducible flow graphs were generated with the confusion-ized machine, and it becomes possible to only remove an ambiguous predicate. Since this is opposed, the alternative conversion sometimes shown in drawing 13 b can be used. When a confusion-ized release machine removes $P^F$ blindly, the code obtained as a result becomes less right.

Especially drawing 13 a and 13b have illustrated the conversion for changing a reducible flow graph into an irreducible flow graph. In drawing 13 a, the loop body S2 is divided into two portions ($S^a_2$** and $S^b_2$), and a false jump is inserted at the beginning of $S^b_2$. Similarly in drawing 13 b, S1 is divided into two partial $S^a_1$ and $S^b_1$. $S^b_1$ is moved into a loop and ambiguous predicate $P^T$ ensures that $S^b$ is always performed in front of a loop body. 2nd predicate $Q^F$ ensures that $S^b_1$ is performed only once.

6.2.4 Depend on the call to a standard library for the program of most which is written in by removal Java of a library call and programming idiom greatly. Since the semantics of a library function is well-known, this call may provide an advantageous cue to a reverse engineer. The reference directions to a Java library class are always based on a name, and a problem gets worse according to the fact that these names cannot be confusion-ized.

In many cases, the confusion-ized machine could oppose this by providing the original version of a standard library simply. For example, the call to a Java (hash table enforcement is used) dictionary class is red although accompanied by the same action. - It may be changed to the call to the class carried out as a black tree. Although the cost of this conversion is not so large about execution time, it is large about the size of a program.

A similar problem also generates Cliche (or pattern) who is the common programming idiom frequently generated with much applications. The reverse engineer of a rich experience jumps over an understanding from ** about the program do not get it used to seeing. - The pattern to apply will be searched in order to start. The linked list in Java™ is taken into consideration as an example. A Java™ library does not have at all a standard class which provides common list operations, such as insertion, deletion, and listing.

Instead, most Java™ programmers will build an object list as a special thing by doubling and linking them on the next field. Repeatedly it lets this list pass, it is the pattern which was dramatically common in the Java™ program. Technology invented in the field of the automated program recognition (Linda Mary Wills, automation program recognition by which refer to it and entailment is carried out to this book; feasibility) [ and ] artificial intelligence and 45(1-2);113 - 172-1990 reference -- referring to it -- it is usable because of discernment of a common pattern, and substitution with not so clear a thing. For example, in the case of a linked list, in an element array, it may be not so common a thing called cursor, and a standard list data structure may be expressed.

6.2.5 Table interpretation One of the most effective (and it is expensive) conversion is a table interpretation. A view converts one Type (this example Java byte code) of a code into a different virtual-machine code. This new code is executed by the virtual-machine interpreter by which entailment was carried out with confusion-ized finishing application at this time. Clearly, two or more interpreters which perform the Type which accepts a respectively different language, and from which confusion-ized finishing application differs may be contained in specific application.

Since the slowdown of a single figure usually exists about each interpretation level, this conversion should be reserved by the Type of the code which constitutes the small portion of

overall run time, or needs protection of a high level dramatically.

6.2.6 Addition of redundant operand If the ambiguous variable of once some is built, an algebra principle can be used in order to add a redundant operand to arithmetic expression. In this way, $U_1$ metric will increase. In the case of the integer expression whose numerical system is not a problem, this technology functions best clearly. In the sentence (1') whose following has been confusion-ized, the ambiguous variable P whose value is 1 is used. In a sentence (2'), a value builds ambiguous subexpression P/Q which is 2. When a sentence (2') is reached clearly, as long as those quotients are always set to 2, it can be made to take a value which is different during program execution at P and Q.
(1) X=X+V;$=^{T}$=> (1') X-X+V*$P^{-1}$;
(2) Z=L+1; (2') Z=L+($P^{=2Q}$/$Q^{=P/2}$)/26.2.7 Parallelization of a code Automatic parallelization is important compiler optimization used for increasing the performance of application of running on a multi-BUROSESSA computer. The Reason which wants us to parallelize a program is various with a natural thing. It is not for our increasing performance, and in order to make actual flows of control ambiguous, it desires to increase parallelism. : with two available operations considered, i.e., 1., -- the dummy process which does not perform a useful task at all can be created. The serial Type of 2. application codes can be divided to the multiplex Type performed in parallel.

When application is running on an unit processor computer, we can expect having an execution time penalty with these great conversion. Since the elasticity of these conversion is high, this can be accepted in many situations. That is, since the number of the execution paths which let a program pass and which are considered becomes large exponentially with the number of execution processes, static analysis of a parallel program is dramatically difficult. Probably, it turns out that :, i.e., a reverse engineer, in which parallelization also produces the effect of a high level again is what a parallel program is harder to understand far compared with a sequential program.

When it does not include data subordinacy at all, 1 code Type is parallelized easily and it deals in it, as shown in drawing 14. For example, these can be run in parallel to the case where $S_1$ and $S_2$ are two data independence type sentences. In the programming language which does not have an explicit parallel construct like a Java™ language at all, a program may be parallelized using the call to a thread (light weight process) library.

A code Type including data subordinacy as shown by drawing 15, By inserting a suitable synchronization unmodified instruction called an away and advance, it may be divided to a concurrency thread (Michael Wolfe by which entailment is carried out to this book as reference, highly efficient compiler .Addison-Wesley for parallel computing, 1996.). Refer to ISBNO-8053-2730-4. Although such a program is run sequentially fundamentally, flows of control will be shifted from one thread to the following thread.

6.3 Set conversion A programmer conquers the peculiar complexity of programming by introducing abstraction. Procedure abstraction is the most important although abstraction exists on many levels of one program. Since it is such, for a confusion-ized machine, it is important to make a procedure and a method call ambiguous, and it obtains. Below, it takes into consideration

about some methods of making a method and a method call ambiguous, i.e., in-line processing, outline processing, interleave, and cloning. The code (since ****** was probably carrying out mutual attribution logically) to which : with the same fundamental idea that is back [ these / all ], i.e., (1) programmer, gathered to one method is divided, The code considered not to belong to that it should distribute over the whole program and both (2) is gathered by the form of one method.

6.3.1 In-line one and outline method In-line processing is compiler optimization important with a natural thing. This is also very useful confusion-ized conversion from removing procedure abstraction from a program. In-line processing is high conversion of elasticity very much from the trace of abstraction not remaining at all in a code, when a procedure call is once replaced by the main part of the called procedure and the procedure itself is removed (this is fundamentally one-way).

.In-line processing of Procedures P and Q is carried out what at the call site, and drawing 16 shows whether next, it is removed from a code.
Outline processing (let me change a series of sentences to a subroutine) is the very useful sister exchange to in-line processing. We create false procedure abstraction by extracting the end of the beginning of the code of Q, and the code of P to the new procedure R.

In the object-oriented language like a Java™ language, in-line processing is not necessarily conversion completely one-way as a matter of fact always. I will consider method call m.P(). The actual procedure called will be influenced by the run time type of m. Calling two or more methods at a specific call site, or when it can do, we do in-line processing of all the methods considered (Jeffrey Dean, all the program optimization of an object-oriented language by which entailment is carried out to this book as reference.). refer to the University of Washington doctoral dissertation and 1996 -- a suitable code is chosen by branching about ma type.

Therefore, even as for after the in-line processing of a method, and removal, the confusion-ized finishing code may include the trace of original abstraction for or a little still more. For example, drawing 17 has illustrated the in-line processing of a method call. Unless the type of m can be determined statistically, all the methods which can tie up m.P() and which are considered call, and in-line processing must be carried out at a site.
6.3.2 Interleave method Detection of an interleave method is an important and difficult reverse engineering task.
Drawing 18 shows whether what we do with the interleave of the two methods declared in the same class. The main part and parameter list of a method are annexed, and a view adds an excessive parameter (or global variable), and discriminates from the call to each method. Ideally, character should be similar so that a method may enable annexation of a common code and a parameter. The case of drawing 18 is this and the 1st parameter of M1 and M2 has the same type here.

6.3.3 Clone method When you are going to understand the purpose of a subroutine, a reverse engineer will inspect the signature and main part with a natural thing. However, different environment where the call of it is carried out is important for understanding the action of a routine similarly. We can make this process still more difficult actually by confusion-izing the

call site of a method to make it it seem that a routine which is different although that is not right is being called.
Drawing 19 can create a different version of the plurality of a method by applying a confusion-ized conversion set which is different in an original code. We use method dispatch for choosing between versions which are different in run time.

Although method cloning is similar with predicate insertion conversion of drawing 11, it differs here in that it is said that it is going to use not an ambiguous predicate but method dispatch for choosing between bar SHON from which a code differs.

6.3.4 It meant improving the performance of loop conversion (especially) numerical value application, and much loop conversion has been designed. About extensive investigation, it is Bacon. Refer to [2]. Since some of these conversion increases complexity metric mentioned above about drawing 7, they are useful for us. Loop blocking as shown in drawing 20 a is used, in order that an internal loop may improve the cash action of a loop by dividing interaction space so that it may fit in cash. The roux PUAN roll reproduces the main part of 1 time or a multiple-times loop as it is shown in drawing 20 b. It is a compile time point, and when the loop boundary is known, on the whole, it can act as Ain Rolle of the loop. Loop division as shown in drawing 20 c is changed to two or more loops accompanied for the loop accompanied by a compound main part by the same repetitive space.

Since all of three conversion increase the sum total code size and the condition number of source application, it increases $U_1$ and $U_2$ metric. Similarly, loop blocking conversion introduces excessive nesting, therefore $U_3$ metric increases it.

It dissociates, and when applied, the elasticity of these conversion is very low. Great static analysis does not need a confusion-ized release machine to roll again the loop which acted as Ain Rolle. However, when conversion is put together, elasticity rises dramatically. For example, when the simple loop of drawing 20 b is given, we apply Ain Rolle first, and division can be applied to the next and, finally we can apply blocking. Probably, analysis of quantity most for a confusion-ized release machine is needed in order to return the loop acquired as a result to the original form.

6.4 Ordering conversion A programmer has the tendency to organize one's source code so that the locality may be made into the maximum. In a similar manner [ two logically related items ] within a source text, when physically near, a program reads a view further, and it becomes easy to understand it. Locality exists between the method in the sentence between the clauses in :, for example, a formula, on which this kind of locality functions on all the levels of a source, and in a basic block, the basic block in a method, and a class, and the class in a file. The spatial locality of all the kinds can provide a useful cue to RE. Therefore, in being possible, we always randomize arrangement of the arbitrary items in source application. About the item (for example, method in a class) of some types, this is trivial. In order to discern which re-ordering is technically effective (for example, sentence in a basic block) in other cases, it is data subordinacy analysis (entailment is carried out to this book as reference (David F.Bacon, Susan L.Graham, and Oliver J.Sharp.)). Compiler conversion A CM Computing Surveys for high performance computation, 26(4):345-December, 420-1994, Highly efficient KOMBAIRA for

http://www.acm.org/pubs/toc/Abstracts/0360-0300/197406.html., and Michael Wolfe. parallel computing. Addison-Wesley and 1996.ISBN 0-8053-2730-4 -- referring to it -- it is carried out. effect that these conversion is low -- having (great ambiguity is not added to a program) -- the elasticity is high and one-way in many cases. For example, when arrangement of the sentence in a basic block is randomized, the trace of an order of a basis will not remain in the code obtained as a result at all.

To "in-line outline" conversion of ordering conversion, the 6th, and Section 3 or 1 especially, it is useful sister exchange. The effect of the conversion may be reinforced carrying out in-line processing of two or more procedure calls within the (1) procedure P, randomizing an order of the sentence in (2) P, and by carrying out outline processing of the contiguity Type of the sentence of (3) P. Thus, according to the inside of false procedure abstraction, it is before put into the unrelated sentence which were a part of several different procedures.
It is also possible by for example making it run backward in some cases to re-order a loop. Such loop inversion conversion is common in a highly efficient compiler. compiler conversion ACM Computing Surveys for David F.Bacon, Susan L.Graham, and Oliver J.Sharp. high performance computation, and 26(4): -- in December, 420-1994 [ 345-].
http.//www.acm.org/pubs/toc/Abstracts/0360-0300/197406.html.

7. Data conversion This section explains the conversion which confusion-izes the data structure used with source application. Such conversion is classified as conversion which performs memory of data, coding, set, or ordering as shown in drawing 2 e.

7.1 Memory and coding conversion In many cases, there is a "natural" method for memorizing the specific data item in a program. For example, assigning the local integer variable of size suitable as a repetitive variable will be chosen in order to repeat each element of one array. although other variable types are usable -- being such -- others -- probably, a variable type is inferior in respect of nature -- carrying out -- probably -- ** -- probably, it is efficiently inferior

The "natural" interpretation of the bit pattern which the specific variable based on the type of the variable can have exists in many cases. For example, generally the 16-bit integer variable which memorizes a BITTOBA turn "0000000000001100" is assumed to express the integral value "12." With a natural thing, these are mere agreements, and other interpretations are possible.

Confusion-ized memory conversion tries to choose an unnatural storage class for dynamic data and a static data. Similarly, coding conversion tends to choose unnatural coding about a common data type. Although memory conversion and coding conversion use them in many cases, combining, each of these conversion is able to be alone used depending on the case.

7.1.1 Change of coding As a simple example of coding conversion, when $c_1$ and $c_2$ are constants, the integer variable i is replaced by $i_0=c_1*i+c_2$. It is possible to choose $c_1$ as involution of 2 for efficiency. If [ the following example ] $c_1=8$ and $c_2=3$,

```
{                          = T =>        {
int i=1;                                 int i=11;
while (i < 1000)                         while (i < 8003)
     . . . A[i] . . .;                     . . A[(i-3)/8] . . .;
    i++;                                      i+=8;
}                                        }
```

It is necessary to cope with the problem of overflow (and the case of a floating point variable accuracy) with a natural thing. It is possible that overflow does not occur because of the range of the variable in question (this range can be determined by [ which use a static-analyses method ] depending especially or questioning a user), and to investigate that it can change into a bigger variable type.

On the other hand, there may be a trade-off between elasticity and cost on the trade-off between elasticity and effect, and another side. A simple mark like $i_0=c_1+i+c_2$ of the above-mentioned example

Although ****** adds only the execution time of a slight addition, a general compiler analysis method (Michael Wolfe.High performance Compilers For Parallel Computing.Add1 son-Wesley and 1996.ISBN O-8053-2730-4 -- and) David F.Bacon and Susan. L. Graham and Oliver. J. Sharp.Compiler transformations for high-performance computing.ACM Computing Surveys, 26(4):345-420, and December. 1994. It is possible for confusion release to be carried out using http.//www.acm.org/pubs/toc/Abstracts/0360-0300/197406.html.

7.1.2 Promotion of variable Some simple memory conversion which promotes a variable exists in a more nearly general-purpose class from the specialized storage class. Although it is generally low, when used combining other conversion, the effect and elasticity of such memory conversion are very effective, and are obtained. For example, in Java, an integer variable is able to be promoted by the integer object. It is applied when the same things are other scalar types with which all have the corresponding class "package-ized." The object will be automatically removed, when an object is no longer referred to already, since Java™ supports arrangement of a garbage. Here, it is the example.

```
 {                                {
 int I=1;                         int i = new int(1);
 while (i < 9)   = T =>           while (i.value < 9)
      ...A[i]...;                      ...A[i.value]...;
      i++;                             i.value++;
}                                 }
```

It is also possible to change the life of a variable. Such the simplest conversion changes a local variable into the global variable shared between the independent procedure calls later. For example, both P and Q are unable to be simultaneously active [ both procedure P and procedure Q ] with reference to a local integer variable (if a program does not contain a thread). this can be asked by investigating a static call graph (static call graph) -- to a case, it is able to make a variable into a global variable and to be shared among these procedure.

```
 void P()  {              int C;

 int i; ...I...           void P()  {

                          ...C...

                           }

 void Q()      = T =>     while (i.value<9)

 int k;...k...            ...C...

}                          }
```

Since the number of the global-data structures referred to by P and Q is increased, this conversion increases $u_5$ metric.

7.1.3 Division of variable A Boolean variable and other variables within the restricted limits are able to be divided into two or more variables. The variable V divided into k variable $p_1$, ..., $p_k$ is described to be V-[$p_1$, ..., $p_k$]. Typically, the effect of this conversion will increase with k. It is common for conversion cost to also increase with k, therefore to restrict k to 2 or 3 regrettably.

In order to make it possible to divide the variable V of Type T into the two variables p and q of Type U, The three information fragments (p;q) F, i.e., the function which carries out the map of the value of p and q to the value of V which carries out (1) correspondence, (2) function g(V) which carries out the map of the value of V to the value of corresponding p and q, and (3) -- it is required to give the new operation (it corresponds to the elementary operation to the value of Type T) by which the cast was carried out from a viewpoint of an operation over p and q. In the remaining portion of this section, V is a Boolean variable type and it is assumed that p and q are small integer variables.
Drawing 21 a shows selection of expression which can be carried out about a division Boolean variable. This table shows that it is equivalent to V being an imitation, then the thing to say, when V is divided into p and q, and when it is p=q=0 or p=q=1 on a certain point of a program. Similarly, p= 0 and q= 1 or p= 1, and q= 0 correspond truly.

When shown by this new expression, the replacement about various nest binary operations (for example, AND, OR) must be devised. One approach is providing the execution-time look-up table about each operator. The table about "AND" and "OR" is independently shown in drawing 21 c and drawing 21 d. If it assumes that two Boolean variable V1= [p, q] and V2= [r, s] are given, $V_1$&$V_2$ will be calculated as AND [2 p+q, 2 r+s].
The result of three division, Boolean variable A= [a1, a2], B= [b1, b2], and C= [c1, c2], is shown in drawing 21 e. In order to calculate the same Boolean expression, I hear that the interesting

side of expression which this invention person chose has some usable methods, and there is. For example, it differs mutually and the sentence (3') and sentence (4') of drawing 21 e are visible, although both assign an imitation to a variable. Similarly, although the sentence (5') completely differs from the sentence (6') mutually, both calculate A&B.

All of the effect of this conversion, elasticity, and costs increase according to the number of a variable by which an original variable is divided. Elasticity is further enhanced by choosing coding at the time of execution. In other words, the execution-time look-up table from drawing 21 b to drawing 21 d, Although not built by compile time (this will make it possible to conduct static analyses to an execution-time look-up table), it is built by the algorithm contained in confusion-ized application. In order to calculate an elementary operation so that this may be performed in the sentence (6') in drawing 21 e with a natural thing, it prevents using an in-line code.

7.1.4 Conversion to procedure data of static data The static data, especially the character string include many practical information useful for a reverse engineer. The method for confusion-izing a static string is changing a static string into the form of the program which generates the string.

A program with possible their being DFA or a try scan (Trie traversal) is able to generate other strings.
For example, I will consider the function G of drawing 22 currently built so that a string "AAA", "BAAAA", and "CCB" may be confusion-ized. The values generated by G are G(1) = "AAA" and G (2) = "BAAAA" and G(3) =G(5) = "CCB" and G (4) (not actually used by program) = "XCB." In the case of other argument values, even if G is completed, it is not necessary to end.

It is not very desirable to gather calculation of all the static string data in the form of a single function with a natural thing. When G function is divided into the form of the small component rather than embedded into the "usual" control flow of a source program, a far advanced effect and elasticity are realized.

It is important to point out that it is possible to combine this technique with table interpretation conversion of Section 6.2.5. I hear that the intention of the confusion-izing changes one section of a Java byte code into the code for another virtual machine, and there is. Typically, this new code will be memorized as static string data within a getting [ confused ]-izing program. However, in order to obtain the effect and elasticity of a still higher level, the above-mentioned string is able to be converted into the program which generates the string as mentioned above.

7.2 Set conversion In contrast with imperative language and a functional language, an object-oriented language is data-oriented rather than being control-oriented. Although in other words control is composed of an object-oriented program at the circumference of a data structure, it is not composed of other methods . -- this means that the significant part of the reverse engineering of object-oriented application is trying restoration of the data structure of a program.

On the contrary, it is important that a confusion-ized machine tends to conceal such a data structure.

In most object-oriented languages, the number of the methods for performing the data aggregate is two, namely, they are a data set in the form of arrangement, and a data set in the form of an object. The following three sections examine the method by which such a data structure can be confusion-ized.

7.2.1 If the combination range of annexation $V_1$ of a scalar variable, ..., $V_k$ suits the accuracy of $V_M$, two or more scalar variable $V_1$, ..., $V_K$ are able to be merged into one form of variable $V_M$. For example, two 32-bit integer variables are able to be merged into one 64 bit variables. The operation to each variable will be changed into the form of an operation over VM. As an easy example, it considers that the two 32-bit integer variables X and Y are annexed to the form of 64 bit-variable Z. Following annexation type $Z(X, Y) = 2^{32}*Y+X$ is used, and the arithmetic identical equation of drawing 23 a is obtained. Some easy examples are shown in drawing 23 b.

It is shown that the two 32-bit integer variables X and Y are annexed by especially drawing 23 to one 64 bit-variable Z. Y occupies 32 bits of Z upper parts, and X occupies 32 bits of bottoms. When one of the actual ranges of X or Y are able to deduce from a program, an intuitive more much more unclear annexation is able to be used. Drawing 23 a shows the rule for the addition and the multiplication by X and Y. Drawing 23 b shows some easy examples. This example can be further confusion-ized by merging the form of "Z+=47244640261", for example (2') (3').

The elasticity of variable annexation is very low. In order to presume that a certain specific variable comprises two annexation variables actually, the confusion release machine should just investigate that the arithmetic operation set is applied to the specific variable. By introducing the bow gas (bogus) operation which cannot be dealt with all of an appropriate operation to each variable, it is possible to increase elasticity. It is possible to insert the operation which seems to annex two half portions of Z, for example by rotation (Z, 5), i.e., $If(P^F)$ Z-rotate, in the example of drawing 23 b.

One modification of this conversion is that $V_1$, ..., $V_K$ are annexed to one arrangement suitable type [ following ].
$V_A$= 1...k $V_1$ ... when $V_kV_1$, ..., $V_k$ are object reference variables, For example, the element types of VA are able to be all the classes which are in a level higher than any of the type of $V_1$, ..., $V_k$ in inheritance hierarchy.
7.2.2 Reconstruction of arrangement Some conversion is able to be devised in order to confusion-ize the operation performed to arrangement. or two or more arrangement is annexed [ whether it divides one arrangement into some secondary arrangement, or ] to one arrangement -- one arrangement -- folding up (a number of dimension is increased) (fold) -- or what is done for flattening of the one arrangement (a number of dimension is decreased) (flatten) is possible. Drawing 24 shows some examples of arrangement reconstruction. The arrangement A is divided into the two secondary arrangement A1 and the form of A2 in a sentence (1-2). A1 holds the element of A which has even indexes, and A2 holds the element which has odd indexes.
It is shown how it is possible for the sentence (3-4) of drawing 24 to be carried out at the integer arrangement B, and for mutual arrangement of the C to be carried out at the form of arrangement BC as a result. The element from the arrangement B and the element from the arrangement C are uniformly distributed covering the whole arrangement acquired as a result.

It is shown how as for a sentence (6-7), it is possible for one-dimensional-array D to be folded up by the form of the two-dimensional array D1. Finally, the sentence (8-9) shows inverse transformation. Flattening of the two-dimensional array E is carried out to the form of the one-dimensional array E1.

Division and folding of arrangement increase $u_6$ data complexity metric. On the other hand, annexation and flattening of arrangement decrease metric one of this. Although these conversion may seem to show having only a small or negative effect as for this, this is easy to produce an error actually. I hear that complexity metric of drawing 7 is not useful to grasp the important side of some data structure conversion, and a problem is in it. That is, such conversion will introduce the structure which did not exist at the beginning, or will remove structure from an original program. This can increase confusion-ization of a program remarkably. For example, the programmer who declares two-dimensional array does so intentionally. A selected structure carries out the map of the data currently operated appropriately anyhow. When the arrangement is folded up by one-dimensional structure, the reverse engineer will be deprived of precious practical information.

7.2.3 In the present object-oriented language like the correction Java™ language of succession relation, main modularization and an abstraction concept are classes. A class is an intrinsically abstract data type which encapsulates data (instance variable) and control (method). A class is described to be C- (V, M), V of a front type is a set which is an instance variable of C, and M is the method.
In contrast with the conventional concept of an abstract data type, two class $C_1$ and $C_2$ succeed with grouping ($C_2$ has an instance variable of type $C_1$) (by adding a new method and instance variable). It is possible for be alike in $C_2$ extending $C_1$ to be built. Succession is described to be $C_2=C_1UC'_2$. Succession of $C_1$ which is the superclass or a parent class will express $C_2$. U operator is a function which combines the new property defined by $C'_2$ and a parent class. It depends for the exact semantics of U on each programming language. In a language like Java, when applied to an instance variable, it is interpreted as a merger, and on the other hand, when U is applied to a method, it is generally interpreted as an override.

According to metric $u_7$, the complexity of class $C_1$ increases according to the depth (distance from a route) in inheritance hierarchy and direct posterity's number. For example, there are two methods with possible making this complexity increase. That is, it is possible to divide one certain class (factor) and to insert the class of a new imitation (bogus) as shown in drawing 25 b as shown in drawing 25 a.

The problem about class factoring is the elasticity. There is nothing that prevents that the class by which the factor was carried out is simply annexed by the confusion release machine. In order to prevent this, generally, factoring and insertion are together put as shown in drawing 25 d. The option to which the elasticity of conversion of these types is made to increase is making to generate a new object about all the introduced classes into a positive thing.

Drawing 25 c shows modification of the class insertion called "fake refactoring." refactoring is a technique (sometimes automatic) for reconstructing the object-oriented program in which the structure has deteriorated (it is taken in by this Description as reference.)William F.Opdyke and.

RalphE.Johnson.Creating. abstract superclasses. by refactoring.In Stan. C. Kwansny and John F.Buck, editors, Proceedings of the 21 st Annual Conference on Computer Science, and page. 66-73, New York, NY, USA, and February 1993.ACM Press.ftp://st.cs.uiuc.edu/pub/papers/refactoring/refactoring-superclasses.ps. I would like to be referred to. RIFAKU Tring is a process of two steps. It is detected that two separate classes carry out same operation to the 1st as a matter of fact seemingly. the feature common to both classes is moved into a new (probably -- ** -- abstract) parent class by the next. Although fake refactoring is the same operation, it is carried out only to two class $C_1$ without common operation, and $C_2$. When both classes have an instance coefficient same type, these classes are moved into new parent class $C_3$. The method of $C'_3$ may be a version with a bug-ridden (buggy) method from $C_1$ and $C_2$.

7.3 ordering conversion in the section 6.4, it is useful to randomize an order that calculation (it is when possible) is performed -- **** -- it was shown that it is-izing. Similarly, it is useful to randomize an order of the declaration in source application.

In particular, in this invention, an order of the method in a class, and an instance variable and the temporary bara meta method of a method is randomized. A actual order of corresponding with a natural thing in the case of the latter must be re-ordered. The effect of such conversion is low and elasticity is a uni directional.

In many cases, it will also be possible to re-order the element under arrangement. When it states briefly, in this invention, vagueness (opaque) coding function f(i) which carries out the map of the i-th element within original arrangement to the new position of the re-ordered arrangement is provided.

```
 {                                        {
int i=1, A[1000];                        int i=1, A[1000];
while (i < 1000)          = T =>         while (i < 1000)
     ...A[i]...;                              ...A[f(i)]...;
     i++;                                     i++;
}                                         }
```

8. Ambiguous value and ambiguous predicate As mentioned above, ambiguous predicates are the main building blocks in the design of the conversion which confusion-izes a control flow. As a fact, the quality of almost all control conversion is directly dependent on the quality of such a predicate.

The section 6.1 showed the example of the simple ambiguous predicate which has trivial and weak elasticity. This means that an ambiguous predicate is able to be decoded using local static analysis or global static analyses (an automatic confusion release machine can discover the value). Far more advanced resistance [ as opposed to / generally / an attack with a natural thing ] is needed. Although worst exponential time is ideally taken to decode it (setting in the size of a program), it is desirable only for polynomial time to be able to build an ambiguous predicate

without **** to build it. This section explains such two techniques. The 1st technique is based on aliasing and the 2nd technique is based on the light weight process (lightweight poess).

8.1 When ambiguous structure aliasing which uses an object and an alias is possible, the static analysis between procedure is always complicated remarkably. In the language which has dynamic allocation, a loop, and an IF statement, it cannot be exact, cannot flow and cannot actually opt for depended type alias analysis.

In this section, in order to build the ambiguous predicate which is low cost and is elastic to an automatic confusion release attack, the difficulty of alias analysis is used.

8.2 Ambiguous structure which uses thread A parallel program is difficult to conduct static analysis compared with a program one by one. The Reason is the interleaving semantics of a parallel program. That is, n sentences in parallel field PAR $S_1$, $S_2$, ..., $S_n$, and ENDPAR are n!

Performing by the method by which individuals differed is possible. In spite of this, some static analyses of a parallel program are able to be conducted in polynomial time [18], and, on the other hand, others are n! It needs to take all interleaving of an individual into consideration.
In Java, a parallel field is built using the light weight process known as a thread. (Seeing from this invention person's viewpoint) A Java thread has the two useful characteristics. Namely, the scheduling policy of (1) Java thread is not strictly specified depending on language specification,

Therefore, it will be dependent on an implementation and the actual scheduling of (2) threads has the characteristic of being dependent on asynchronous events like the asynchronous events generated by the user interaction, and network traffic. When combined with the peculiar interleaving semantics of a parallel field, this means that it is very difficult to conduct the static analyses of the thread.

In this invention, in order to generate the ambiguous predicate (drawing 32 should be referred to) which needs worst exponential time for a decipherment, the above-mentioned consideration result is used. This fundamental idea is dramatically similar with the idea currently used with the section 8.2. That is, it is maintained by the state where an ambiguous inquiry is able to be performed, although the global-data structure V is generated and it is updated infrequently. I hear that V is updated by the thread under present execution, and there is a point of difference.

It is possible for it to be a dynamic data structure like the dynamic data structure by which V was generated by drawing 26 with the natural thing. Into the component of each of the thread, a thread will move the global pointers g and h at random by performing the call for movement and insertion in asynchronous. This has the advantage of combining data competition with the interleaving effect and the aliasing effect, in order to obtain very high elasticity.
In drawing 27, V uses the farther simple example which are one pair of global integer variables X and Y, and is illustrating the above-mentioned idea. In the case of the arbitrary integers x and the integer y, this is based on the publicly known fact from the basic number theory [ "1 / $7y^2$-" ] that it is not equal to $x^2$.
9. confusion release and prevention conversion this invention person's confusion-ized conversion -- many (especially control conversion of the section 6.2) -- it is possible to be described as

embedding the bow gas program (bogus program) in a real program. In other words, the real program which getting [ confused ]-izing application comprises actually two programs merged into one, namely, performs a useful task, and the bow gas program which calculates useless information are merged into one. The only purpose of this bow gas program is to confuse a potential reverse engineer by concealing a real program behind an unrelated code.

The above-mentioned ambiguous predicates are the main mechanisms which a confusion-ized machine can use freely, in order to prevent it from a bow gas inner program being identified easily and removed. For example, in drawing 28 a, a confusion-ized machine embeds the bow gas cord (bogus code) protected by the ambiguous predicate into three sentences of a real program. The task of a confusion release machine is investigating getting [ confused ]-izing application, identifying an internal bow gas cord automatically and removing it. In order to perform this, a confusion release machine must evaluate the structure, after identifying an ambiguous structure first. This process is shown in drawing 28 d from drawing 28 b. Drawing 29 shows the structure of a semi-automatic confusion release tool. This tool has taken in some publicly known techniques in the community of reverse engineering.

The remaining portion of this section examines some of such techniques briefly, and in order to make confusion release into difficulty more, in it, various countermeasures (what is called prevention conversion) which can use a confusion-ized machine are explained.

9.1 The prevention conversion explained above in relation to prevention conversion drawing 2 g completely differs in effect from control conversion or data conversion. In contrast with control conversion or data conversion, the main targets of prevention conversion are not covering a program to human being's reader. Rather, prevention conversion should make difficulty more the publicly known automatic confusion release technique (original prevention conversion).
Or it is designed spy out the known problem in a present confusion release machine or decompiler (target prevention conversion).

9.1.1 Originally originally [ prevention conversion ], prevention conversion generally has a low effect and high elasticity. The most important thing is having the capability prevention conversion enhancing the elasticity of other conversion originally. As an example, it is assumed that it is what ordering is re-finished so that a "for" loop may be performed to an opposite direction as suggested with the section 6.4. The Reason it was possible to have investigated that a loop does not have loop conveyance data subordinacy to this conversion was able to be applied. What prevents conducting the analysis with confusion release machine same with a natural thing, and returning a loop to forward direction execution does not have anything. In order to prevent this, it is possible to add the bow gas data subordinacy over a reverse loop.

```
 {                                {
for(i=1;i<-10;i++)  -T->              int B[50];
     A[i]-i                           for(i=10;i<=1;i--)
}                                     A[i]=i;
                                      B[i]+=B[i*i/2]
                                 }
```

It depends on the complexity of bow gas subordinacy, and the state of the art of subordinacy analysis for the elasticity which prevention conversion applies to loop re-ordering conversion originally [ this ] [36].
9.1.2 Target prevention conversion As an example of target prevention conversion, A HoseMocha program is considered (Mark D.LaDue.HoseMocha.http://www.xynyx.demon.nl/java/HoseMocha.java, January1997). This program, Mocha deconstructivism BAIRA (Hans Peter.) In order to investigate the weak point of VanVliet.Mocha---The Java decompiler.http://web.inter.nl.net/users/H.P.van.Vliet/mocha.html and January 1996. It is designed specially. A HoseMocha program inserts a special command after all the return sentences in all the methods in a source program. Although this conversion does not have influence at all to operation of application, it is enough for making Mocha crash.

9.2 Discernment and evaluation of ambiguous structure The most important portion of confusion release is discernment and evaluation of an ambiguous structure. Please care about that discernment and evaluation are separate activities. an ambiguous structure -- being local (contained in a single basic block) -- or -- being global (contained in a single procedure) -- or what is been between procedure (it is distributing over the whole program) is possible. if (x*x== ($7^{F}$*y*y-1)) is a local ambiguous predicate -- R=X*X; ...;S=7*y*y-1; ...;if (R==$S^{F}$) ... is a global ambiguous technical term. Probably, a structure is an ambiguous structure between procedure, when calculation of R and S is performed in a mutually different procedure. With a natural thing, discernment of a local ambiguous predicate is easier than discernment of the ambiguous predicate during procedure.

9.3 Discernment by Bataan collation The confusion release machine can use the knowledge of SUTORAREJI used with a known confusion-ized machine, in order to identify an ambiguous predicate. The designer (by decompiling a confusion-ized machine) of a confusion release machine, Or it is possible to build the pattern matching rule [ it is possible to investigate a confusion-ized machine and ] which can identify the ambiguous predicate generally used by investigating simply the getting [ confused ]-izing code which a confusion-ized machine generates. This method will work most effectively to a simple local predicate like x*x== (7*y*y-1) or random($1^{F}$, 5) <0.
In order to block the trial of pattern matching, the confusion-ized machine must avoid use of an obligatory ambiguous structure. It is also important to choose an ambiguous structure similar to the structure used with real application on syntax.

9.4 Discernment by program slicing A programmer will generally discover that an understanding is more difficult than a program with a getting [ confused ]-izing version of a program original for a reverse engineer. The "fruit" with which the main Reason (a) Is valid in the getting [ confused ]-izing program

I hear that the code snippet which the dead bow gas cord was inlaid into the code, and was connected with (b) logic target is disassembled, and it distributes over the whole program, and it is. The Rybar engineer is able to use a program slicing tool, in order to oppose such confusion-ization. Such a tool can assist a dialogue with a reverse engineer, in order to decompose a program into the form of manageable chunk (chunck) called a slice. The slice of the program P about the point p and the variable v comprises all the sentences of P which can be contributed to the value of v in p. Therefore, the program slicer can extract the sentence of the algorithm which calculates the ambiguous variable v from a getting [ confused ]-izing program, even when a confusion-ized machine makes the whole program distribute such a sentence.
In order to use a slicing as the discernment tool in which validity is more inferior, some usable strategies exist with a confusion-ized machine. Add parameter alias A parameter aliases are two temporary bara meta (or temporary bara meta and a global variable) which refers to the same memory location. The strict slicing between procedure increases according to the number of the potential alias in a program, and, on the other hand, the number of this potential alias increases exponentially according to the number of a formal parameter. Therefore, when a confusion-ized machine adds to a program the dummy parameter by which aliasing was carried out. The confusion-ized machine exerts coercion so that a slicer is decelerated remarkably, or a slicer may be made to produce an un-strict slice (when a high-speed slicing is needed) (when a strict slice is required).
Unravel (and David W. James R.Lyle, Dolorres R.Wallace, James R.Graham, Kelth B.Gallagher, Joseph P.Poole) Binkley.Unravel:A CASE. tool to assist evaluation. of high integrity software.Vo1ume 1:Requirements and design.Technical Report NIS-TIR 5691 and U.S.Departmentof. Although Commerce and addition variable subordinacy (add variable dependency) like the slicing tool which has generally spread like August 1995 function suitable for calculation of a small slice, Excessive time may be required to calculation of a larger slice. For example, when it was used to the C program of 4000 lines, there was a case which needed time to exceed for 30 minutes for Unravel completing slice calculation. In order to pull out such a feature compulsorily, a confusion-ized machine must increase slice size by adding bow gas variable subordinacy. In the following example, on appearance, although it contributes to the value of x, the size x of slice calculation is increased by adding two sentences which do not contribute actually.

```
main()          {          =T =>          main()  {
int x=1;                                  int x=1;
x - x * 3;                                     if (PF ) x++;
}                                         x = x + V0 ;
                                          x = x * 3;
                                          }
```

9.5 Static analyses The confusion release machine can use a getting [ confused ]-izing program, in order to analyze the result of all predicates. Thus, the arbitrary confusion release methods which investigate the feature at the time of execution of getting [ confused ]-izing application, i.e., static analyses, will be called.

Since it may become clear that the predicate which returns the always same truth value in many trial runs is an ambiguous $P^T$ ($P^F$) predicate, attention of the reverse engineer to all the predicates that return such a truth value is urged to a confusion release machine. Although the confusion release machine cannot replace such a predicate recklessly by truth (imitation), this is because doing so may be too dangerous. Although many applications contain the "true" predicate which inspects the conditions generated only under an exceptional situation, probably, for a confusion release machine, such application seems to operate similarly to an ambiguous predicate. for example, pif (LeapYear) -- please consider ...

Static analyses can still be used also for evaluation. When the potential ambiguous predicate (for example, $P^T$) within the program M is already identified, that value (truth) is presumed and version M' of the getting [ confused ]-izing program which the above-mentioned ambiguous predicate is this point estimate, and has already been replaced is created. M and M' is performed in parallel to the same input, and in order to check that M and M' produces the same output, comparison is performed to the next. When these outputs are the same, it is possible to conclude that the predicate was a part of bow gas which is not true application as shown in drawing 30.

It must be secured that the selected input covers appropriately all the paths in the above-mentioned program. This will be difficult when similarly the above-mentioned program includes the path (if (Leap Year) ...) which are not rare. When it is dramatically difficult to generate much right inputs / output data and especially the internal structure of application is strange, or this is dramatically difficult when an input is performed through a complicated (it is alike occasionally, it carries out and applied to a Java program -- as) graphical user interface.

In order to prevent discernment by static analyses, a confusion-ized machine may choose from the conversion which inserts a $P^T$ predicate or a $P^F$ predicate so that priority may be given to the conversion which inserts a P(as [ show / in drawing 11 b ]) ? predicate.

The countermeasures over static analyses in which another adoption is possible are some predicates' being decoded simultaneously, and designing an ambiguous predicate so that there may be nothing, if it is ******. One of the methods for performing this is giving side effects to an ambiguous predicate. In the following example, a confusion-ized machine determines that sentence $S_1$ and $S_2$ must perform only the always same number of times (a kind of static flow analysis). These sentences are confusion-ized by introducing the ambiguous predicate which is a call to function $Q_1$ and function $Q_2$. Function $Q_1$ and function $Q_2$ fluctuate a global variable.

{ =T => {

$S_1$; int k=0;

```
S2;                          bool Q1 (x)     {
}                            k+=2^31 ; return (P^T 1)}
                             bool Q2 (x)     {
                             k-=2^31 ; return (P^T 2)}
                            {
                             if (Q1 (j) T ) S1 ;
                                  . . .
                             if (Q2 (k) T ) S2 ;
                             }
```

k will overflow, when a confusion release machine tends to replace one predicate (they are not both) by truth. As the result, the program by which confusion release was carried out will be completed by an error condition.

9.6 Evaluation by data flow analysis Confusion release is similar to code optimization various type. if (False) -- it is the dead code (dead code) deletion to remove ...
It is the code hoisting (code hoisting) to move the same code from if sentence brunch (for example, $S_1$ and $S_0^1$ in drawing 28), and these are general code optimization techniques.

If an ambiguous structure finishes being identified, it will become possible to try evaluation of the structure. In a simple example, it comes out enough by the constant propagation which uses an arrival definition data flow analysis (reaching definition data-flow analysis), and a certain thing is possible. x=5;...;y=7;...;if(x*x==(7*y*y=1)....

9.7 Evaluation by theorem proving When not powerful as enough for a data flow analysis to decode an ambiguous predicate, it is possible to try for a confusion release machine to use theorem proving. It is dependent on the capability of the theorem proof program of a state of the art (a check is difficult), and the complexity of a theorem with required being proved whether this is unable to be possible.The theorem (for example, $x^2(x+1)^2=0 \pmod 4$) which can be proved by induction with a natural thing is in the range of access of the present enough theorem proof program.

In order to make a matter still more difficult, it is possible to use the theorem in which a known proof about the theorem for which it is known that proof is difficult, or it does not exist. In the following example, probably, it must prove that the bow gas loop (bogus loop) is always completed, in order to investigate that a confusion release machine is the code (live code) in which $S_2$ was useful.

```
{                    =T =>          {
S1;                                 S1;
S2;                                 n = random(1, 2^32);
}                                   do
                                        n = ((n%2)!=0)?3*n+1:n/2
                                    while (n>1);
                                    S2;
                                     }
```

This is known as a Collatz problem. It is guessed that the above-mentioned loop will always be completed. Although the known antecedent basis of this guess does not exist, it is known that that code will be completed about all the numbers to $7*10^{11}$. Therefore, this confusion-izing is safe (similarly the confusion-ized original code completely operates), and it is difficult to perform confusion release.

9.8 Confusion release and partial evaluation Confusion release is similar also to partial evaluation further. A partial evaluation machine divides a program into two portions, i.e., the static portion by which precomputation can be carried out with a partial evaluation machine, and the dynamic portion performed at the time of execution. Probably, a dynamic portion is equivalent to the original program which is not confusion-ized. A static portion is equivalent to a bow gas inner program, when identified, it is this bow gas inner program at the confusion release time, and it can be evaluated and removed.
Partial evaluation tends to be influenced by aliasing like all other static internal procedure mode analytic methods. Therefore, the same prevention conversion as the prevention conversion which made reference in relation to the slicing is applied also to partial evaluation.

10. Confusion-ized algorithm Next, based on the confusion-ized machine architecture of the section 3, the definition of the confusion-ized quality of the section 5, and explanation of various confusion-ized conversion of the section 6 to the section 9, furthermore it is based on one of the enforcement aspects of this invention, a detailed algorithm is explained.

The loop of the highest level of a confusion-ized tool can have the following general structure. WHILE NOT Done(A) DO S: =SelectCode (A); T:=SelectTransform (S); A:=Apply(T, S);END; SelectCode returns the following source-code object which should be confusion-ized. SelectTransform returns the conversion which must be used in order to confusion-ize this specific source-code object. Apply applies conversion to a source-code object, and updates application according to it. Done determines the time of finishing reaching confusion-ization of a necessary level. It will depend for the complexity of these functions on the sophistication of a confusion-ized tool. It is possible to terminate a loop, when SelectCode and SelectTransform return random source-code object / conversion simply and the size of application crosses the limit where Done is specific as a result of simple determining what number a figure represents.

As for such operation, usual is insufficient.
The algorithm 1 gives description of a code confusion-ized tool including the selection operation and end operation which were refined still farther. In one of the enforcement aspects, this algorithm uses some data structures and such a data structure is built by the algorithms 5, 6, and 7.

When it is $P_S$ about each source-code object S, $P_S$ (S) is the language structure set which the programmer used by S. It is used in order that $P_S$ (S) may discover the suitable confusion-ized conversion about S.

When it is A about each source-code object S, A(S) = $\{T_i$ -- $>V_1$; ...;$T_n$=->$V_n\}$ is mapping to value $V_{[\text{ from conversion Ti ] i}}$, and describes how it is appropriate to apply $T_i$ to S. Since specific conversion introduces the new code "is unnatural and has" this idea for S, I hear that such conversion may be unsuitable about the specific source-code object S, and in S, a certain new code of this looks the not the best, therefore it will be easy to discover it to a reverse engineer.

Probably, the more the code introduced by conversion $T_i$ suits fitness more, the more felicity value (appropriateness value) $V_i$ is large.
When it is I about each source-code object S, I (S) is a confusion-ized priority of S. It is described whether it is important only for which that I (S) confusion-izes the contents of S.

Probably I (S) is HIGH when S contains important trade secrets, and, on the other hand, it is mainly "bread and butter."
I (S) will be LOW when a code is included.
When it is R about each routine M, R (M) is an execution time rank of M.
In order that much time may perform M rather than which other routines, when being spent, it is R(M) =1.

The primary input to the algorithm 1 is set $\{T_1;T_2$ of the application A and confusion-ized conversion.; ... It is$\}$. Further, this algorithm also requires the information about each conversion, and requires three (numerical value is returned although it is the same as that of function of same name in section 5) quality-functions $T_{res}$ (S), $T_{pot}$ (S), $T_{cost}$ (S), and function $P_t$ especially.
When applied to the source-code object S, $T_{res}$ (S) returns the measure of the elasticity (that is, which does T bear appropriately at the attack from an automatic confusion release machine?) of the conversion T.
When applied to the source-code object S, $T_{pot}$ (S) returns the measure of the effect (that is, after being confusion-ized by T, does human being understand however difficult S?) of the conversion T.
$T_{cost}$ (S) returns the measure of the execution time added to S by T, and a space penalty.
$P_t$ carries out the map of each conversion T to the set of the language structure which T will add to application.

The point 3 loads the application which should be confusion-ized from the point 1 of the algorithm 1, and a suitable in-house-data structure is built. The point 4 builds $P_S$ (S), A (S), I (S), and R (M). The point 5 applies confusion-ized conversion until it finishes reaching a necessary confusion-ized level, or until the maximum execution time penalty is exceeded. Finally,

application A' with the new point 6 is rewritten.
Algorithm 1 (formation of code confusion)
Input: Application A C1;C2; which comprises an a source code or an object code file. ...
b) Standard library L1;L2; defined by language ...
c) a confusion-ized conversion set -- {-- T1;T2; ...}.
d) Mapping Pt which gives the language structure set which T will add to application about each of the conversion T.
e) Three functions $T_{res}$ (S) expressing the quality of the conversion T to the source-code object S, $T_{pot}$ (S), $T_{cost}$ (S).
f) Input data set I={I1;I2 to A; ...}.
g) Two numerical value AcceptCost>0 and ReqObf>0. AcceptCost is the measure of the greatest additional execution time / space penalty that a user will accept. ReqObf is the measure of the quantity of confusion-izing demanded by the user.
Output: Getting [ confused ]-izing application A' which comprises a source code or an object code file.
1. application $C_1$;$C_2$; which should be confusion-ized -- load ... Confusion-ized machine, it is possible to load (a) source code file, In this case, it is ** [ probably a confusion-ized machine must contain the perfect compiler front end which conducts a lexical analysis, syntax analysis, and a semantic analysis ] (purely the powerless confusion-ized machine which carries out self-limitation only to syntax conversion). processing without a semantic analysis is possible -- or It is possible to load (b) object code file, and when an object code holds most or all of information in a source code (like [ Java class file / of a case ]), this gentleman method is preferred.
2. library code file L1;L2; referred to directly or indirectly by application -- load ...
3. Build the internal expression of application. It depends for selection of an internal expression on the structure of the source language which a confusion-ized machine uses, and the complexity of conversion. A typical data structure set may contain the following.
(a) The control flow graph about each routine in A.
(b) The call graph about the routine in A.
(c) The succession graph about the class in A.
4. It is [ use) and ] I (S) about mapping R(M) and the $P_s$(S) (algorithm 5.
Use) is built for the (algorithm 6 and use) is built for the A(S) (algorithm 7.
5. Apply confusion-ized conversion to application. In each step, the suitable conversion T which should be applied to the source-code objects S and S which should be confusion-ized is chosen. This process is completed, when a necessary confusion-ized level is reached, or when permissible execution time cost is exceeded.
REPEAT S :=SelectCode(I);
T :=SelectTransform(S,A);
T is applied to S and the suitable data structure from the point 3 is updated.;
UNTIL Done(ReqObf,AcceptCost,S,T,I).
6. Reconstruct getting [ confused ]-izing source code BUJIEKKUTO in new getting [ confused ]-izing application A'.
Algorithm 2 (SelectCode)
Input: Confusion-ized priority mapping as calculated by the algorithm 6.
Output: Source-code object S.
I carries out the map of each source-code object S to I (S), and it is this I (S).
It is that measure only with which [ important ] to confusion-ize ** and S. In order to choose the

following source-code object which should be carried out [ **** ]-izing, I is able to be processed as a priority encoder. In other words, S is chosen so that I (S) may be maximized.
Algorithm 3 (SelectTransform)
Input: a source-code object S.
b) Felicity mapping A as calculated by the algorithm 7.
output: -- the conversion T -- in order to choose the most suitable conversion for applying to a certain specific source-code object S, the heuristics (heuristics) of the arbitrary number is usable. However, there are two important problems which should be taken into consideration. The conversion chosen as the 1st must be mixed with the remaining portion and nature of a code in S. This is A (S).
Handling is possible by giving priority to the conversion which is alike, sets and has a high felicity value. To the 2nd, it is high. "only value of balancing expenses (bang-for-the-buck) " -- it is desirable to give priority to the conversion to obtain (that is, confusion-ization of a high level is obtained by a low execution time penalty). This is realized by choosing the conversion which minimizes cost at the same time it maximizes effect and elasticity. Such heuristics is incorporated in following code and w1 in this code, w2, and w3 are implementation definition constants.
T -- The conversion T is returned so that >V may be in A (S), and
$(w1*T_{pot}t(S)+w2*T_{res}(S)+W3*V)/T_{cost}$ (S) is maximized.
Algorithm 4 (Done)
The residual level of input aReqObf and confusion-izing.
b) The permissible execution time penalty which AcceptCost(s) and remains.
c) Source-code object S.
d) Conversion T.
e) Confusion-ized priority mapping I.
Output: ReqObf of which renewal of a was done.
b) Updated AcceptCost.
c) Updated confusion-ized priority mapping I.
d) The Boolean returned value which is truth when the terminating condition is reached.
A Done function fills two purposes. As for this function, the source-code object S has finished being confusion-ized.
And in order to make the fact that the reduced priority value must be received reflect, priority-encoder I is updated.
This reduction is based on the combination of the elasticity of conversion, and effect. Further, Done updates ReqObf and AcceptCost and investigates whether the terminating condition is reached. w1, w2, w3, and w4 are implementation definition constants.
I. (S): =I. (S)-. $(w_2T_{pos}.)$ (S) -- $+w_2T_{res}(S)$;ReqObf:=ReqObf-$(w_2T_{pos}(S)+w_2T_{res}$ (S));
AcceptCost:=AcceptCost-$T_{cost}$ (S); . RETURN AcceptCost -- < -- =0 OR ReqObf<=O. algorithm 5 (pragmatic information)
Input a application A.
b) Input data set I={I1;I2 to A; ...}.
Output: MABBINGUR (M) which gives the execution time rank of M about each routine M in aA.
b) Mapping Ps (S) which gives the set of the language structure used by S about each source-code object S in A.
Pragmatic information is calculated. It will be used in order that this information may choose the

proper type of the information about each individual source-code object.
1. Calculate dynamic pragmatic information (that is, application is performed under the profiler based on input data set I provided by the user). R (M) and (the execution time rank of M) about each routine/basic block which show where application spends the greater part of the time are calculated.
2. Calculate static pragmatic information $P_S$ (S). $P_S$ (S) provides the statistics about the kind of language structure which the programmer used by S.
FOR S: Set of the operator which each source-code object O:=S in =A DO uses;
C: = set of the high level language structures (a WHILE sentence, an exception, a thread, etc.) which S uses;
L: Set of the library class / routine which =S refers to;
Ps(S)-OUCUL;
END FOR.
Algorithm 6 (confusion-ized priority)
Input: a application A.
b) The rank of R (M) and M.
Mapping I (S) which gives the confusion-ized priority of S about each source-code object S of output A.

A user is able to provide I (S) clearly, or it is possible to be calculated using the heuristics based on the static data collected with the algorithm 5. The usable heuristics may be as follows.
1. Make I (M) in inverse proportion about the arbitrary routines M of A to the rank of M, i.e., R, (M). That is, this idea is "when spending much time in performing a routine, ** and M will probably be the important procedure which must be confusion-ized severely."
2. Let I (S) be the complexity of S as one of the software complexity measures of the table 1 defines. Also in this case, similarly, rather than the case where the direction of a complicated code is a simple code, I hear that intuitive (it may be wrong) discernment has a high possibility that important trade secrets are included, and it has it.
Algorithm 7 (felicity of confusion-izing)
Input: a application A.
b) Mapping Pt which gives the set of the language structure which T will add to application about each of the conversion T.
c) Mapping Ps (S) which gives the set of the language structure used by S about each source-code object S of A.
Output: Mapping A (S) which gives the felicity of T to S about each source-code object S of A, and each conversion T.
The felicity set A (S) is calculated about each source-code object S. Mapping is fundamentally based on the static pragmatic information calculated with the algorithm 5.
FOR S: Degree of the similarity of a between to each source-code object FOR T:= each conversion DO V:=Pt (T) and Ps (S) in =A DO;
A(S):=A(S)U{T-->V};
An END FOREND FOR11. outline and consideration this invention person recognizes that operating in a different form could probably permit the program with an original getting [ confused ]-izing program in many situations. In particular, most confusion-ized conversion of this invention makes working speed of a target program slow compared with the original program, or it enlarges program size. In this invention, when, and a target program is even

enabled to have different side effects from an original program or an original program is completed by error condition, a target program is even enabled not to end. The only necessary condition of confusion-ized conversion of this invention is that the operation (operation which is experienced by the user) which can observe these two programs must only be the same.
It is a fresh very stimulative idea to make possible such weak identity between an original program and a getting [ confused ]-izing program. Although various conversion is provided and mentioned above, other various conversion will be clear to a person skilled in the art, and it is possible to be used in order to realize confusion-ization for the software security enhancement by this invention.
In order to discover the conversion which is not yet known, a possibility that future various surveillance study will be performed is large. It is expected that surveillance study of the next field will be performed especially.
1. New confusion-ized conversion should be discovered.
2. The correlation between various conversion and ordering should be studied. Ordering of an optimization conversion sequence of this is the same as that of the research in code optimization which is an always difficult problem.
3. The relation between effect and cost should be studied. Knowing which conversion will give only the best "value of balancing expenses" (namely, effect highest, by the minimum execution overhead) about the code of each kind is called for.
Please refer to drawing 31, in order to survey all the above-mentioned conversion. Please refer to drawing 32, in order to survey the above-mentioned ambiguous structure. However, this invention must not be limited to above-mentioned typical conversion and ambiguous structure.
11.1 Confusion-ized capability Encryption and program confusion-ization are extremely similar mutually. It not only concealing information from the eyes of human beings who try to spy out a secret, but performing this concealment in limited time has intention of these both. The enciphered document has the limited storage life. Only in the limitation which does not make it possible to decode frequently the message about the key length as whom progress of the processing speed of hardware was chosen, the enciphered document is [ in / the limitation to which the enciphered program itself is equal to the attack ] only safe. It is applied that it is the same also as getting [ confused ]-izing application. Only in the limitation by which the confusion release machine powerful enough is not yet built, getting [ confused ]-izing application is only safe.

Probably, this is not a problem when evolving application, as long as the time between releases is shorter than the time which a confusion release machine takes for a confusion-ized machine to catch up. The application is already outdated by the time of becoming possible to carry out confusion release of the application automatically, therefore, probably, it has been no longer an object of a competition partner's concern, even if a confusion release machine catches up with a confusion-ized machine.

However, when application contains the trade secrets considered to continue over a some times release, such trade secrets must be protected by means other than confusion-izing. Although the partial server side execution (drawing 2 (b)) is the obvious selection, there is a fault that the execution speed of application serves as a low speed, or execution becomes impossible (when network connection is downed).

11.2 Use of everything but confusion-izing It is interesting to point out that other potential confusion-ized uses may exist in addition to the confusion-ized use as above-mentioned. One possibility is using confusion-ization in order to pursue the pirate edition manufacturing-and-selling contractor of software. For example, a vendor creates the new getting [ confused ]-izing version of its application for each new customer (by introducing a random nature element into a SelectTransform algorithm (algorithm 3)). A getting [ confused ]-izing version which is different to each of the same application is able to be generated. Different seed to a random number generator holds record of the customer who produces a separate version and who sold each version. Probably, this will only be appropriate only when the application is sold and rationed through the network. When a vendor discovers that the pirate edition of its application is sold, it being required for this vendor is only getting the copy of that pirate edition version and discovering who having purchased that original application, comparing it with a database. In practice, it is unnecessary to save Coby of all the sold getting [ confused ]-izing versions. It is enough just to save the sold random number seed.

The pirate edition manufacturing-and-selling contractor of software may use confusion-ization (inaccurate). Since the Java confusion-ized machine outlined on these Descriptions operates on the level of a byte code, what prevents that a pirate edition manufacturing-and-selling contractor performs confusion-ization to the Java application purchased lawfully does not have anything. After that, this getting [ confused ]-izing version is able to be resold. When a pirate edition manufacturing-and-selling contractor faces a lawsuit, it is also possible to claim that they sell actually the version which he does not resell the application purchased first (the codes completely differ despite the join office!), and redesigned lawfully.

Conclusion In conclusion, this invention provides the method and equipment which are realized on [ for blocking, even if the reverse engineering of software is prevented or it is small ] a computer. This can realize execution time or program size as a sacrifice, and on a level with a detailed program changed as the result, although it operates in a different form, it is thought that the method of this invention brings about high usefulness in a suitable situation. In one of the enforcement aspects, the changed program is seen with a non-changing program and carries out the same operation in a top. Therefore, this invention makes possible such weak identity between an original program and a getting [ confused ]-izing program.
although the indication of this invention has been performed mainly in the context called inhibition of the reverse engineering of software -- the water marking (watermarking) of a software object (application is included) -- other applications [ like ] are assumed. This uses characteristic character potentially [ all single confusion-ized procedure ]. A vendor will create a separately different getting [ confused ]-izing version to each customer of an application sale place. When a pirate edition copy is discovered, the vendor can only compare the pirate edition version with an original confusion-ized information database, and can pursue original application.

The specific confusion-ized conversion currently explained in this Description is not comprehensive. It may be used in the new confusion-ized tool architecture of this invention, providing another confusion-ization of a type.
publicly known in the above-mentioned explanation -- etc. -- although the element or integer which has **** has been mentioned, ****, such as having carried out like this, is contained in

this invention as explained separately above.
Although this invention has been explained in the form of a mere example with reference to specific enforcement aspect, it must be understood that change and an improvement are able to be made without deviating from the range of this invention.

---

[Translation done.]

* NOTICES *



1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

---

## DRAWINGS

---

[Drawing 1]

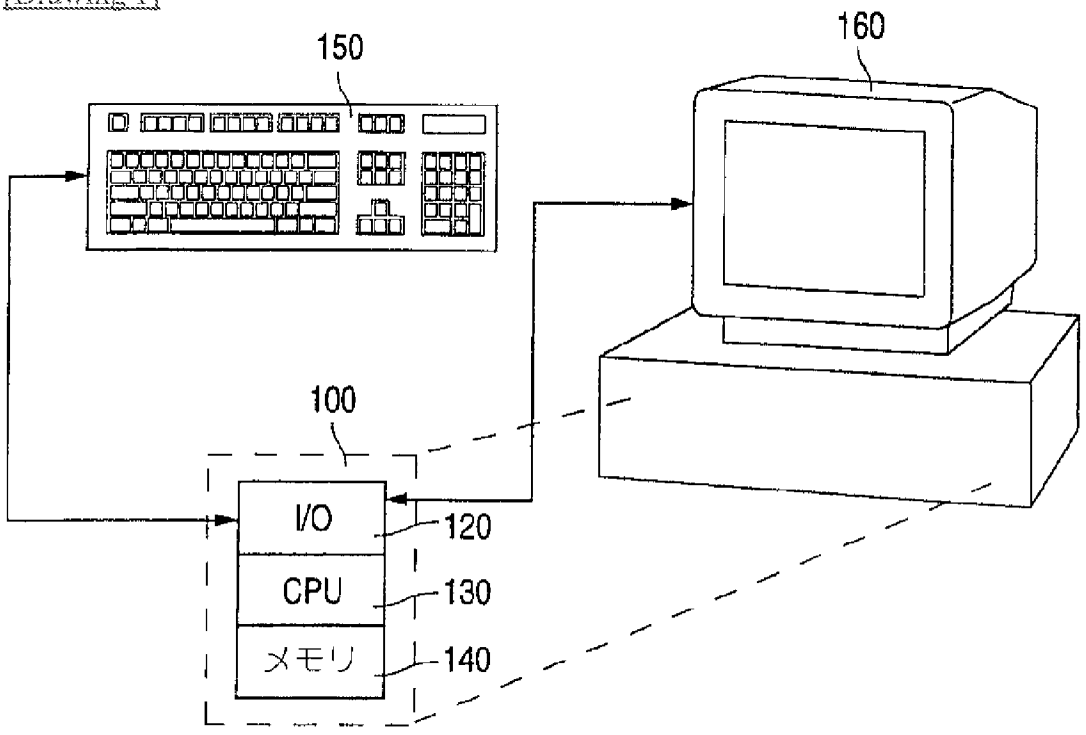



FIG. 1

[Drawing 2]

FIG. 2a

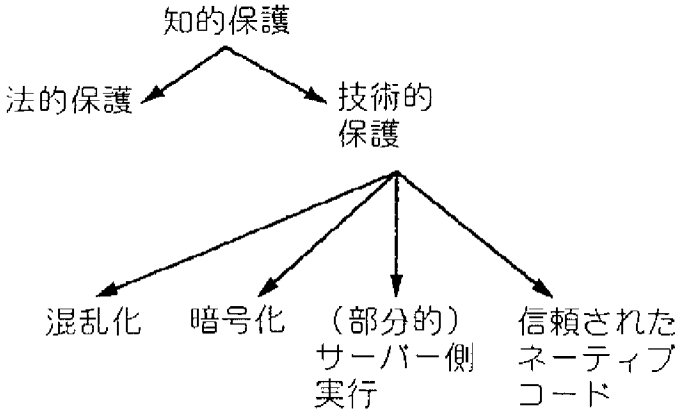
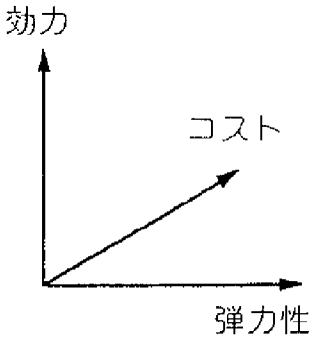



FIG. 2b




[Drawing 2]

FIG. 2c

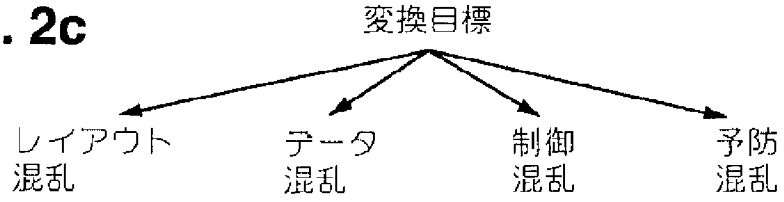



FIG. 2e

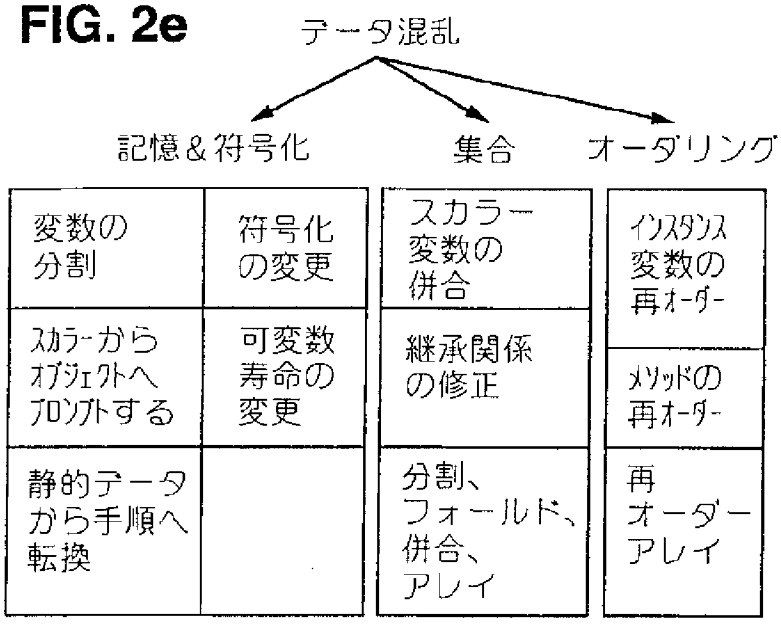
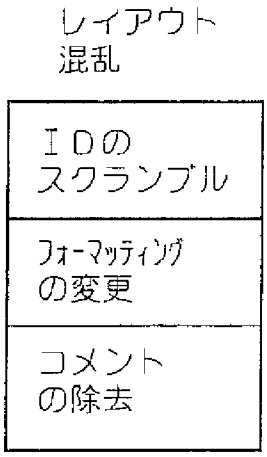



FIG. 2d



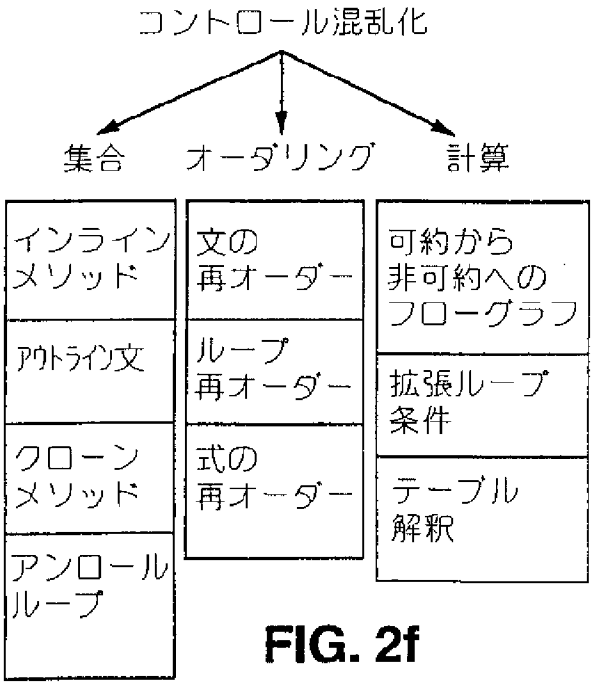




FIG. 2f

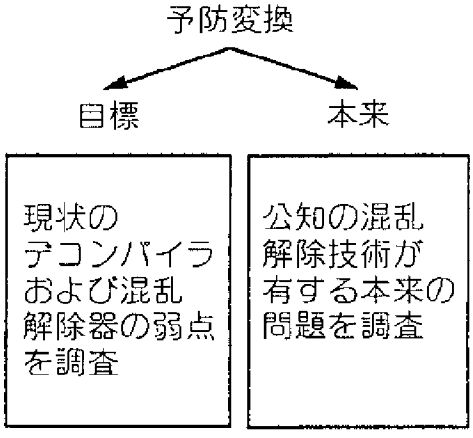



FIG. 2g

[Drawing 3]

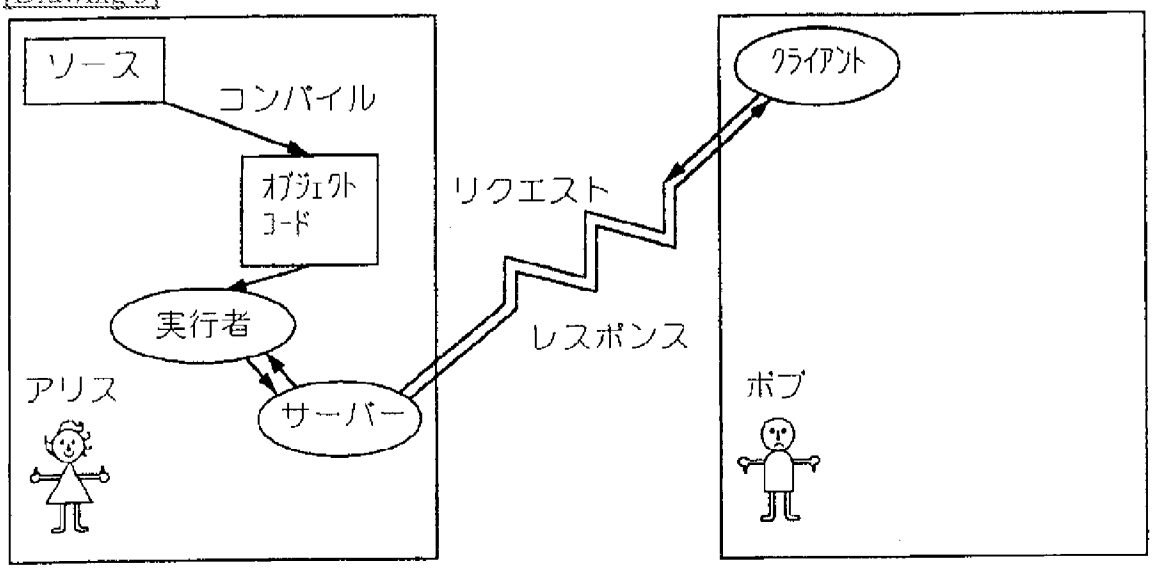



FIG. 3a

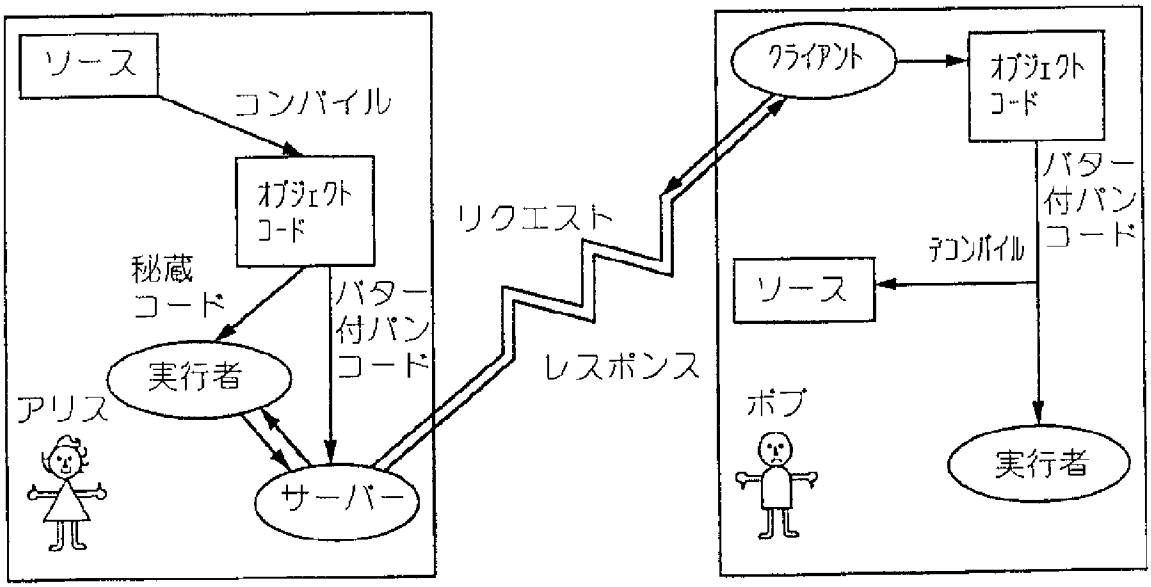



FIG. 3b

[Drawing 4]

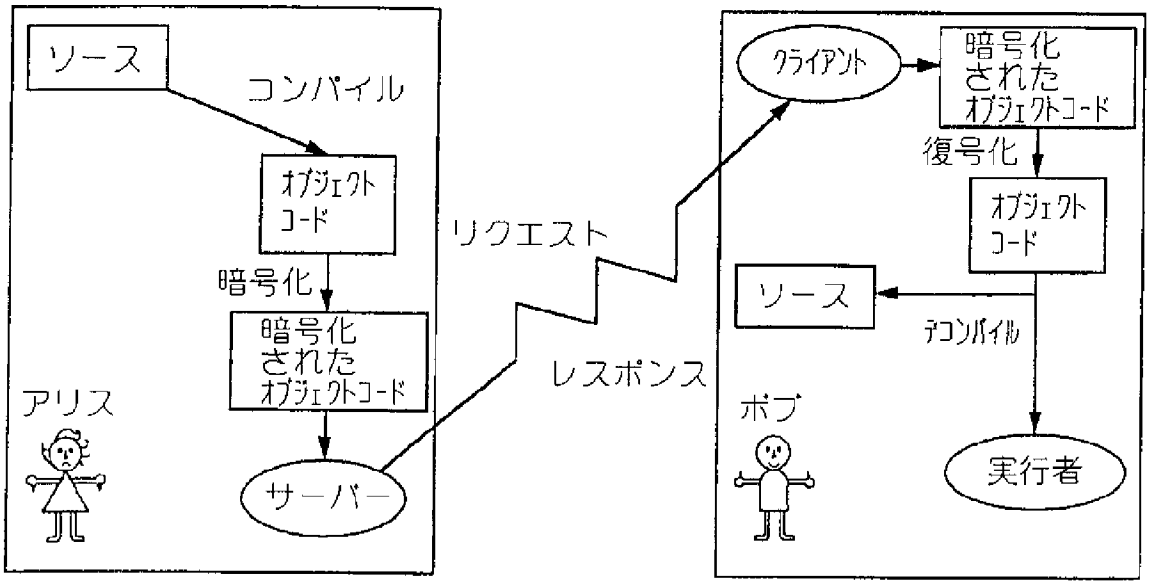



FIG. 4a

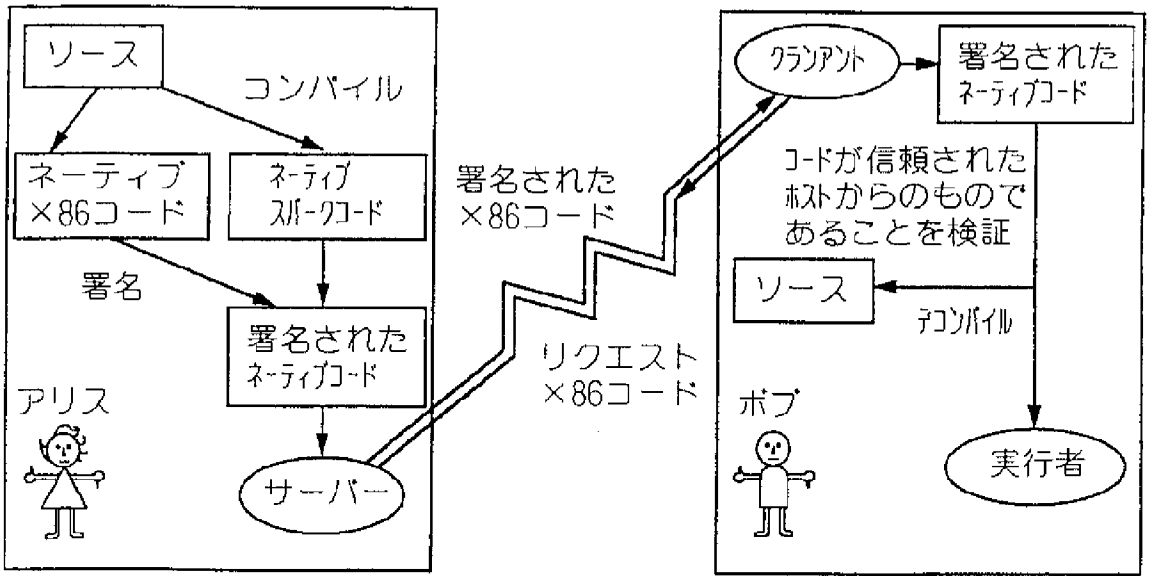



FIG. 4b

[Drawing 5]

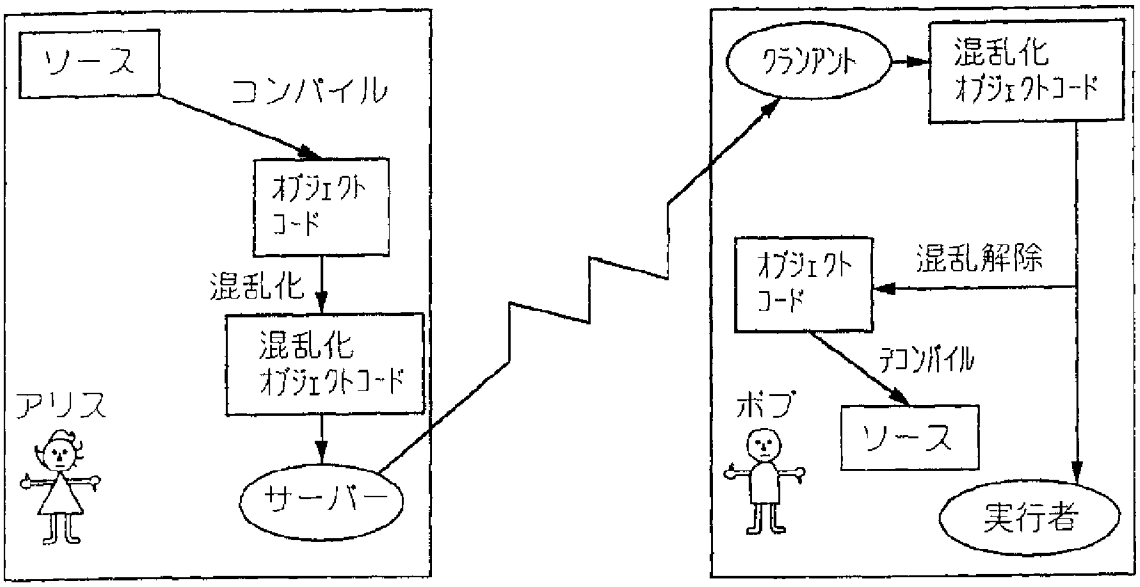



FIG. 5

[Drawing 6]

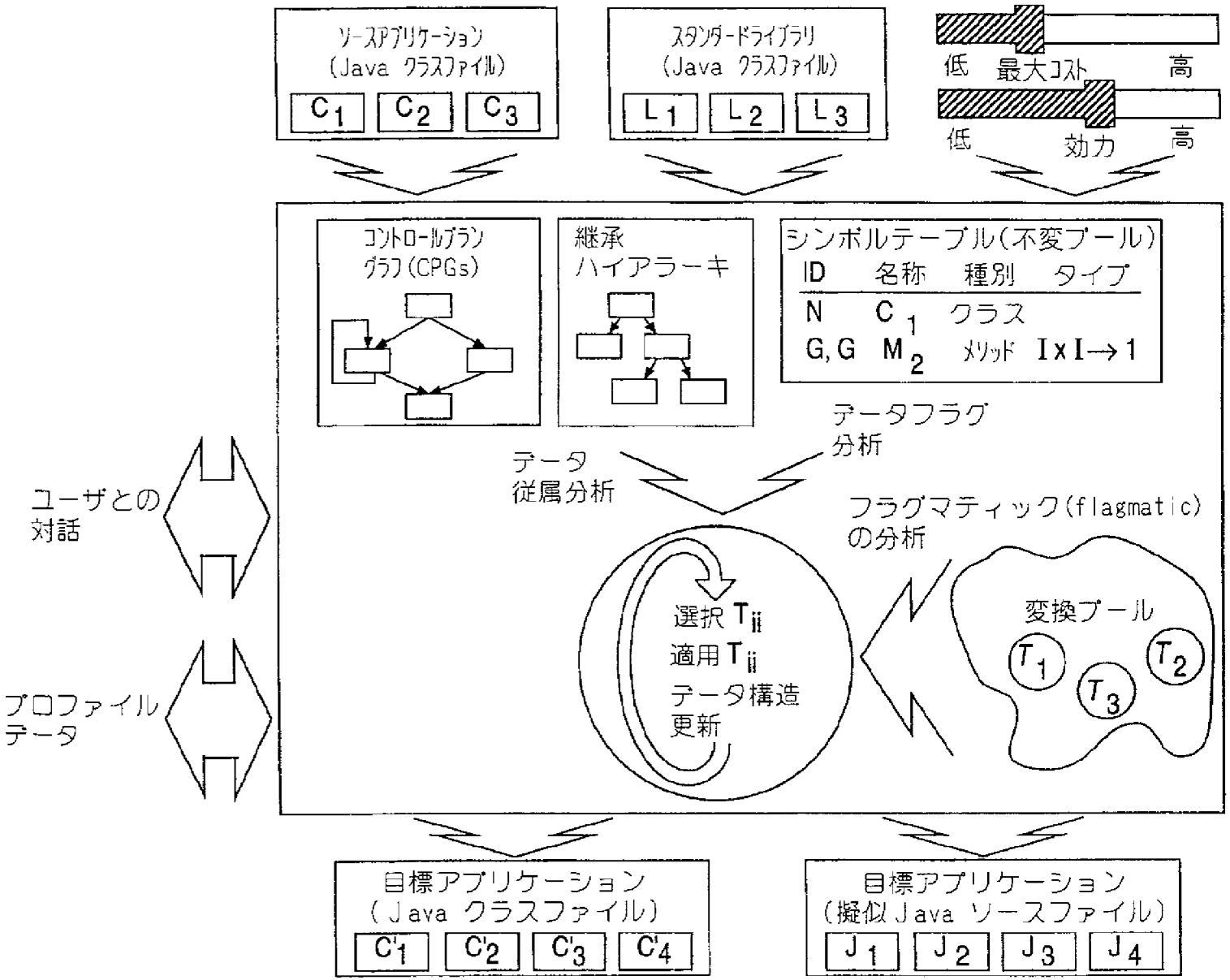

ソースアプリケーション
(Java クラスファイル)
C1 C2 C3
スタンダードライブラリ
(Java クラスファイル)
L1 L2 L3
低 最大コスト 高
低 効力 高
コントロールフラン
グラフ(CPGs)
継承
ハイアラーキ
シンボルテーブル(不変プール)
ID 名称 種別 タイプ
N C1 クラス
G,G M2 メソッド IxI→1
データフラグ
分析
データ
従属分析
ユーザとの
対話
フラグマティック(flagmatic)
の分析
選択 Tii
適用 Tii
データ構造
更新
変換プール
T1
T2
T3
プロファイル
データ
目標アプリケーション
(Java クラスファイル)
C'1 C'2 C'3 C'4
目標アプリケーション
(擬似 Java ソースファイル)
J1 J2 J3 J4

FIG. 6

[Drawing 7]

| メトリック | メトリック名称 | 引用 |
|---|---|---|
| $\mu_1$ | プログラム長 | Halstead |
| | E(P)は、P内のオペランドおよび演算子の数と共に増大する | |
| $\mu_2$ | ｻｲｸﾛﾏﾃｨｯｸ(cyclomatic)の複雑性 | McCabe |
| | E(F)は、F内の述語の数と共に増大する | |
| $\mu_3$ | 入れ子の複雑性 | Harrison |
| | E(F)は、F内条件の入れ子レベルと共に増大する | |
| $\mu_4$ | データフローの複雑性 | Oviedo |
| | E(F)は、F内の内部基本ブロック可変レファレンスの数と共に増大する | |
| $\mu_5$ | ファンーイン／アウトの複雑性 | Henry |
| | E(F)は、Fへの形式パラメータの数と共に、そして、Fにより読まれまたは更新されるグローバルデータ構造の数と共に、増大する | |
| $\mu_6$ | データ構造の複雑性 | Munson |
| | E(P)は、P内にて宣言された静的データ構造の複雑性と共に、増大する。スカラー変数の複雑性は不変である。アレイの複雑性は、ディメンション数と共に、そして、エレメントタイプの複雑性と共に、増大する。レコードの複雑性は、そのフィールドの数および複雑性と共に、増大する。 | |
| $\mu_7$ | OOメトリック | Chidamber |
| | E(C)は、$C_i$内のメソッド数($\mu_7^a$)と、継承木構造内のCの深さ(ルートからの距離)($\mu_7^b$)と、$C_i$のダイレクトサブクラスの数($\mu_7^c$)と、Cが結合される他のクラスの数($\mu_7^d$)と、$C_i$のオブジェクトへ送られたメッセージに応答して実行可能なメソッド数($\mu_7^e$)と、インスタンス変数の同一セットをCのメソッドが参照しない度合($\mu_7^f$)と共に、増大する。注：$\mu_7^f$は結合度、すなわち、モジュールのエレメントがどれ位強く関連しているか、を測る。 | |

*2つのクラスは、もしその一方が他方のメソッドまたはインスタンス変数を使用するならば、結合される。

FIG. 7

[Drawing 8]

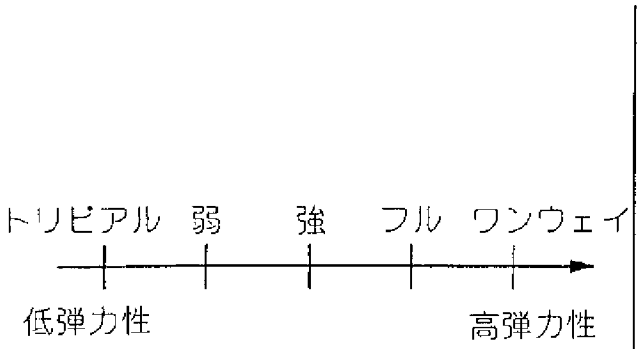



FIG. 8a

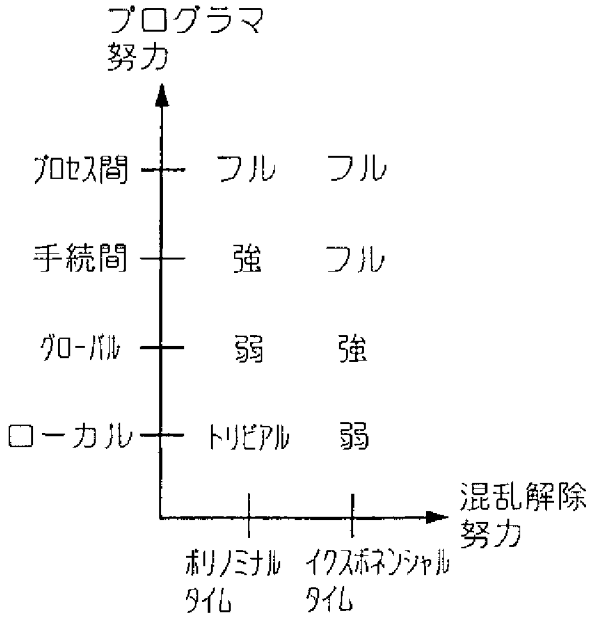



FIG. 8b

[Drawing 9]

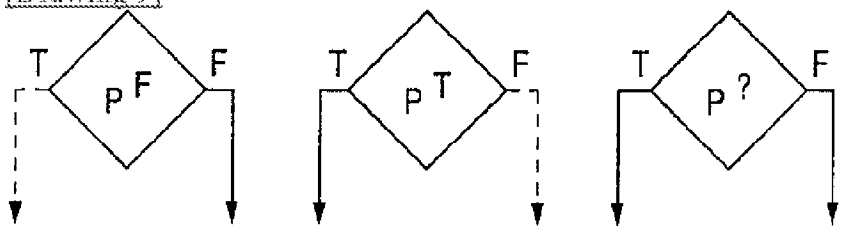



FIG. 9

[Drawing 10]

```
{     int v, a=5; b=6;
      v=11= a + b;
      if (b > 6) T ...
      if (random (1,5) < 0) F...
}
```

FIG. 10a

```
{     int v, a=5; b=6;
      if (...) ...
            ⋮      (b is unchanged)
      if (b < 7) T  a++1
      v=11= (a > 5) 7v=b=b; v=b
}
```

FIG. 10b

[Drawing 11]

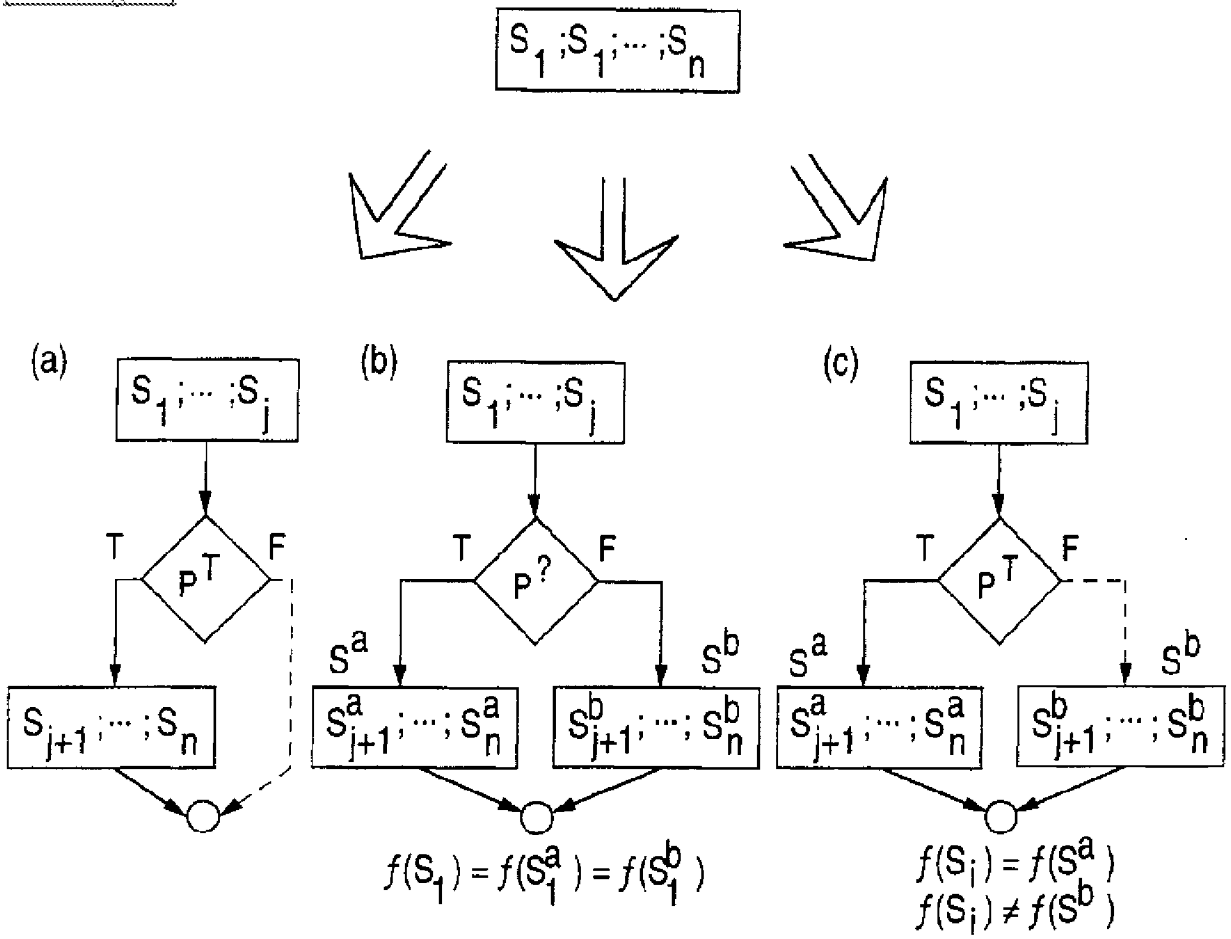
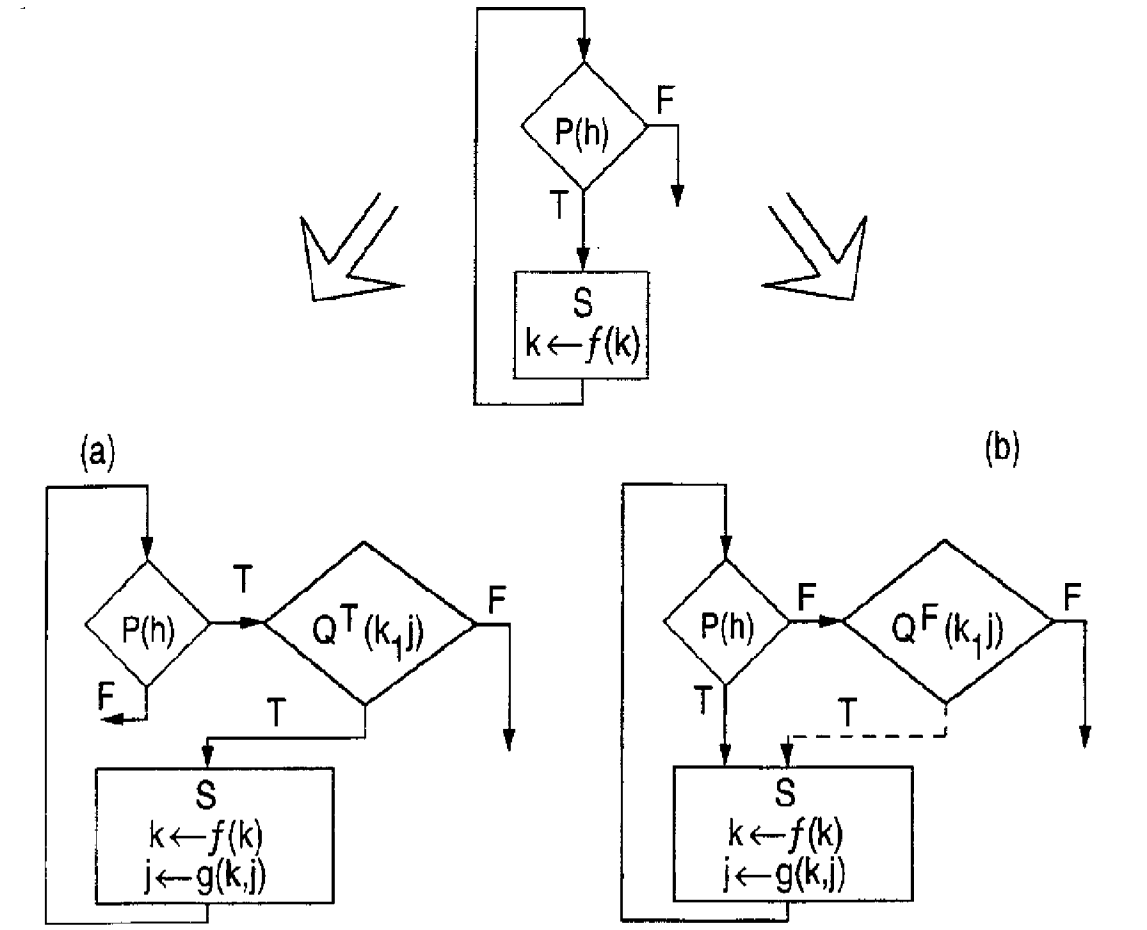



**FIG. 11**

[Drawing 12]




```
i=1
while (1<100) {
        ...
    i++;
}
```

(c)

$\tau$

```
i+1; j=100;
while ((i<100) ff (j* j * (j+1)* (j + 1)%Δ == 0)^T ) {
        ...
    i++;
    j = j * 1 + 3;
}
```

(d)

**FIG. 12**

[Drawing 13]

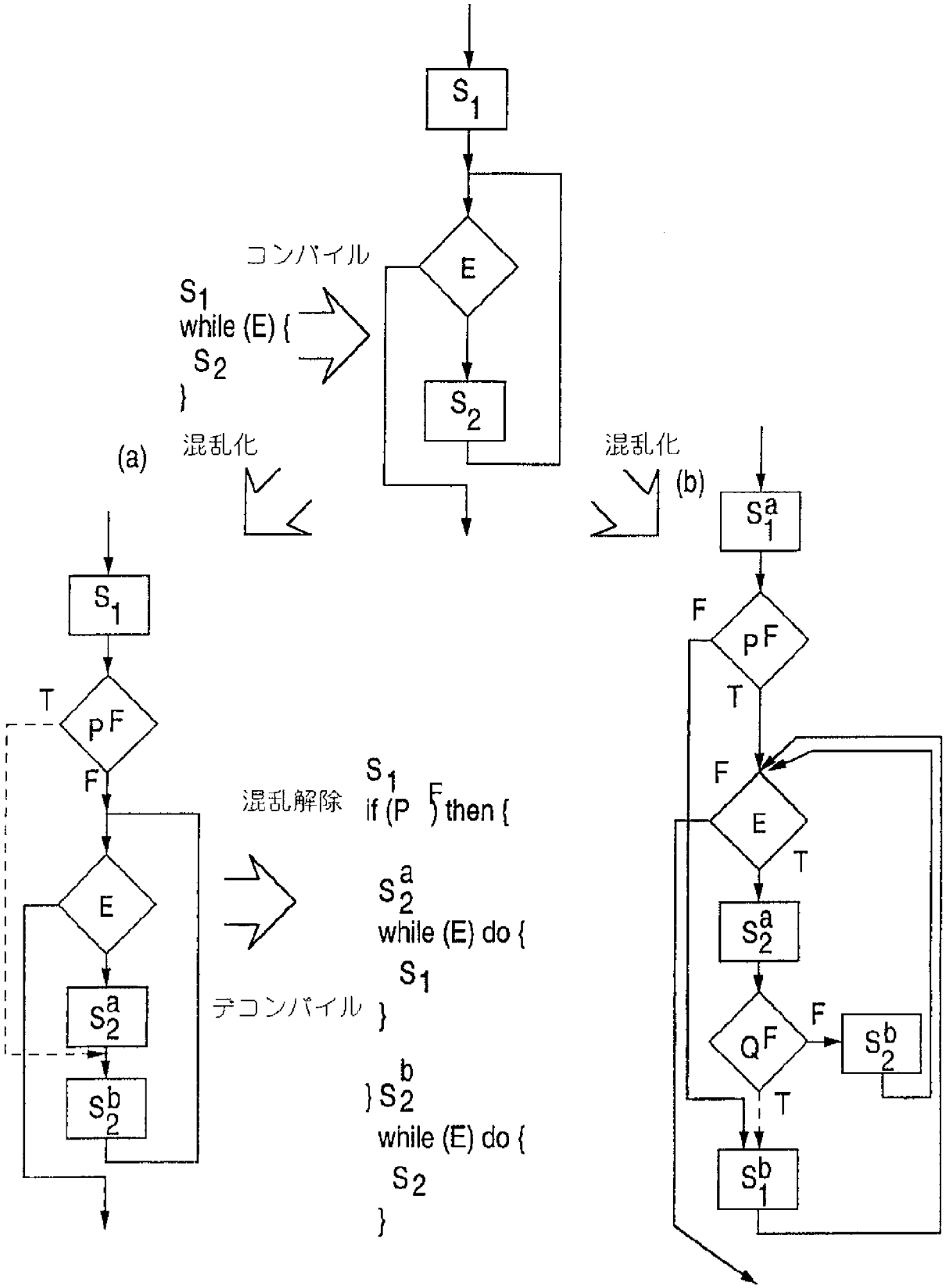

コンパイル
S1
while (E) {
S2
}
E
S2
(a)
混乱化
混乱化
(b)
S1
T
P
F
F
E
S2a
S2b
混乱解除
デコンパイル
S1
if (P ) then {
F
S2a
while (E) do {
S1
}
S2b
while (E) do {
S2
}
S1a
F
P
F
T
F
E
T
S2a
Q
F
F
S2b
T
S1b

FIG. 13

[Drawing 14]

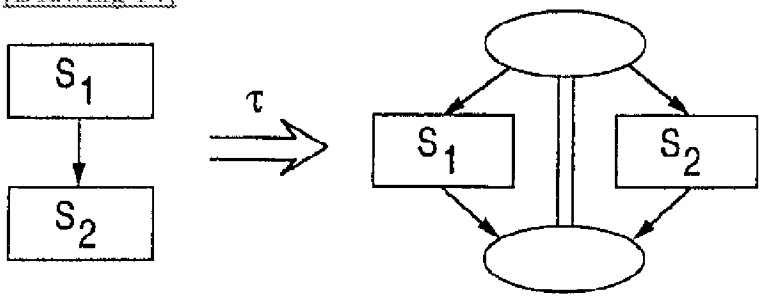



FIG. 14

[Drawing 15]

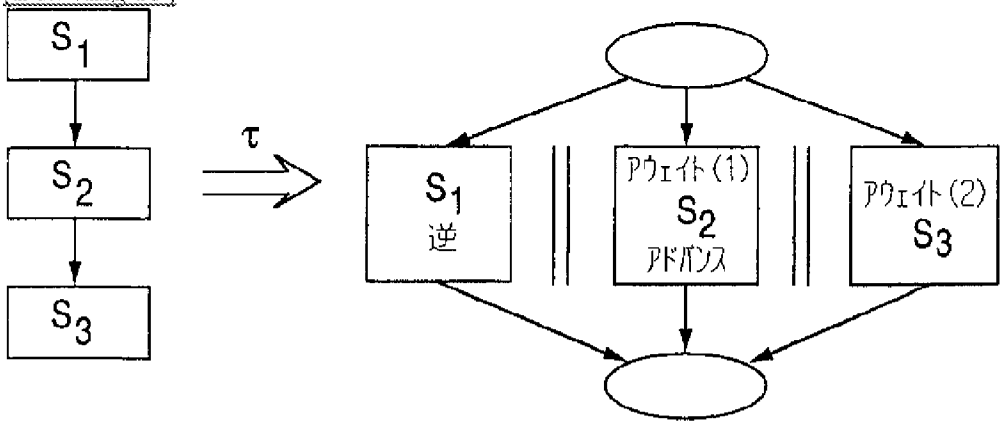



FIG. 15

[Drawing 16]

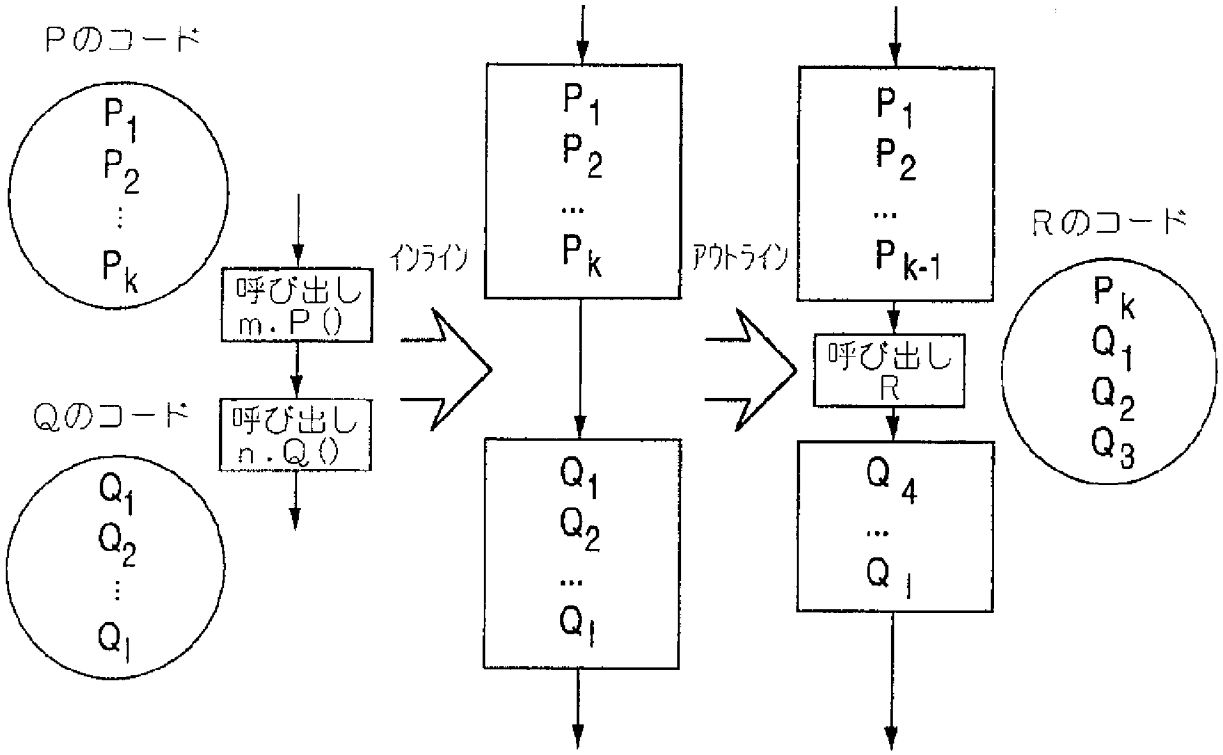



FIG. 16

[Drawing 17]

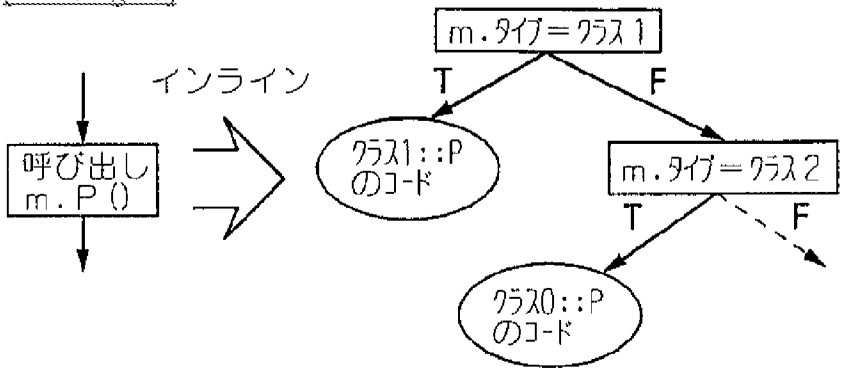



FIG. 17

[Drawing 18]

```
class C {
   method M1  (T1 a)  {
      S 1 M1 ;... S k M1 ;
   }
   method M2 (T1 b; T2 c) {
      S 1 k1 ;... S m k2 ;
   }
}

{  C x=new C;
   x.M1(a); x.M2(b, c); }
```

$\tau$ ⟹

```
class C' {
   method M (Ti a; T2 c; int V) {
      if (V==p) {S 1 M1 ;...S k M1 ;}
      else      {S 1 M1 ;...S m k2 ;}
   }
}

{  C' x=new C';
   x.M(a, c, V=p);
   x.M(b, c, V=g); }
```

**FIG. 18**

[Drawing 19]

```
class C {
   method m (int x)
      {S 1 ... S k}
}

{  C x = new C;
   x.m(8); ... x.m(7);
}
```

$\tau$ ⟹

```
class C1 {
   method m (int x)
      {S 1 a ;...S n a }
   method m1 (int x)
      {S 1 c ;...S n c }
}
class 02 inherits C1 {
   method M (int x)
      {S 1 b ... S k b }
}

{  C1 x ;
   if (P7 ) x=new C1 else x=new C2;
   x.m(5); ...; x.m1(7);
}
```

**FIG. 19**

[Drawing 20]

**FIG. 20a**

```
for (i=1,i<=n,i++)                    τ    for(I=1, I<=n, I+=64)
  for (j=1, j<=n,j++)                          for (J=1,J<=n,J+=64)
    a[1,j]=b[j,i]                                for (i=I,i<=min(I+63,n),i++)
                                                   for (j_J,j<=min(J+65,n),j++)
                                                     a[i,j]=b[j,i]
```

**FIG. 20b**

```
for(i=2,i<(n-1),i++)                  τ    for (i=2,i<(n-2),i+=2) {
  a[i] +=a[1-i]==[i+1]                         a[i] += n[i-1]=a[i+1];
                                               a[i+1] += a[i]=a[i+2];
                                        };
                                        if (((n-2) % 2) == 1)
                                          x[n-1] += a[n-2]=a[n]
```

**FIG. 20c**

```
for(i=1,i<n,i++) {                    τ    for (i=1,i<n,i++)
  a[i] += c;                                   a[i] += c;
  x[i+1]=d+x[i+1]=a[i]                       for (i=1, i<n, i++)
}                                              x[i+i] <d+x[i+1]=a[i]
```

[Drawing 21]

## FIG. 21a

| g(V) p | g(V) q | f(p,q) V | 2p + q |
|---|---|---|---|
| 0 | 0 | False | 0 |
| 0 | 1 | True | 1 |
| 1 | 0 | True | 2 |
| 1 | 1 | False | 3 |

## FIG. 21b

| VAL[p,q] | | p = 0 | p = 1 |
|---|---|---|---|
| q | 0 | 0 | 1 |
| | 1 | 1 | 0 |

## FIG. 21c

| AND[A,B] | | A = 0 | A = 1 | A = 2 | A = 3 |
|---|---|---|---|---|---|
| B | 0 | 3 | 0 | 0 | 0 |
| | 1 | 3 | 1 | 2 | 3 |
| | 2 | 0 | 2 | 1 | 3 |
| | 3 | 3 | 0 | 0 | 3 |

## FIG. 21d

| OR[A,B] | | A = 0 | A = 1 | A = 2 | A = 3 |
|---|---|---|---|---|---|
| B | 0 | 3 | 1 | 2 | 3 |
| | 1 | 1 | 1 | 2 | 2 |
| | 2 | 2 | 2 | 1 | 1 |
| | 3 | 0 | 1 | 2 | 0 |

```
(1)   bool A,B,C;
(2)   A = True;
(3)   B = False;
(4)   C = False;
(5)   C = A & B;
(6)   C = A & B;
(7)   C = A | B;
(8)    if (A) ...;
(9)    if (B) ...;
(10)  if (C) ...;
```

T

```
(1')  short a1,a2,b1,b2,c1,c2;
(2')  a1=0; a2=1;
(3')  b1=0; b2=0;
(4')  c1=1; c2=1;
(5')  x=AND[2*a1+a2,2*b1+b2]; c1=x/2; c2=x%2;
(6')  c1=(a1 ^ a2) & (b1 ^ b2); c2=0
(7')  x=OR[2*a1+a2,2*b1+b2]; c1=x/2; c2=x%2;
(8')  x=2*a1+a2; if ((x==1) || (x==2) ...;
(9')  if (b1 ^ b2) ...;
(10') if (VAL[c1,c2]) ...;
```

## FIG. 21e

[Drawing 22]

```
String G (int n) {
    int i=0,k;
    String B;
    while (i) {
        L1:  if (n==1) {S[i++]="A";k=0;goto L6};
        L2:  if (n==2) {S[i++]="B";k= -2 ;goto L6};
        L3:  if (n==3) {S[i++]="C";goto L8};
        L4:  if (n==4) {S[i++]="K";goto L9};
        L5:  if (n==5) {S[i++]="C";goto L11};
             if (n>12) goto L1;
        L6:  if (k++<=2) {S[i++]="A";goto L6} else goto L8;
        L8:  return S;
        L9:  S[i++]="C"; goto L10;
        L10:  S[i++]="B"; goto L8:
        L11:  S[i++]="C"; goto L12;
        L12:  goto L10;
    }
}
```

FIG. 22

[Drawing 23]

FIG. 23a

$$Z(X + r,Y) = 2^{32} \cdot Y + (r + X) = Z(X,Y) + r$$
$$Z(X,Y + r) = 2^{32} \cdot (Y + r) + X = Z(X,Y) + r \cdot 2^{32}$$
$$Z(X \cdot r,Y) = 2^{32} \cdot Y + X + r = Z(X,Y) + (r - 1) \cdot X$$
$$Z(X,Y \cdot r) = 2^{32} \cdot Y \cdot r + X = Z(X,Y) + (r - 1) \cdot 2^{32} \cdot Y$$

FIG. 23b

```
(1) int X=45, Y=95;          (1') long Z=167759066119551045;
(2) X += 5;          τ       (2') Z += 5;
(3) Y += 11;         ⟹      (3') Z += 47244640256;
(4) X *= c;                  (4') z += (c-1)* (Z & 4294967295);
(5) Y *= d;                  (5') Z += (d-1)* (Z & 18446744069414584320);
```

[Drawing 24]

```
(1)  int A[9];
(2)  A[i] = ...;
       ...
       ...
(3)  int B[8],C[19];
(4)  B[i] = ...;
(5)  C[i] = ...;
       ...
(6)  int D[9]
(7)  for(i=o;i<=B;i++)
       D[i]=2 * D[i+1];
       ...
       ...
       ...
(8)  int E[2,2];
(9)  for(i=Q;i<=2;i ++)
       for(j=0;i<=2;i++)
         swap(E(i,j], E[j,i]);
```

τ ⟹

```
(1')  int A1[4],A2[4];
(2')  if ((i%2)==0) A1[i/2] =...
      else A2[i/2]=...;
        ...
(3')  int BC[20];
(4')  BC[3*i] = ...;
(5')  BC[i/2*3+1+i%2] = ...;
(6')  int D1[1,4];
(7')  for(j=0;j<=1;j++)
        for(k=0;k<=4;k++)
          if (k==4)
            D1[j,k]=2*D1[j+1,0];
          else
            D1[j,k]=2*D1[j,k+1];
        ...
(8')  int E1[8]
(9')  for(i=0;1<=8;i++)
        swap(E[i], E[3=(i%3)+i/3]);
```

**FIG. 24a**

[Drawing 24]

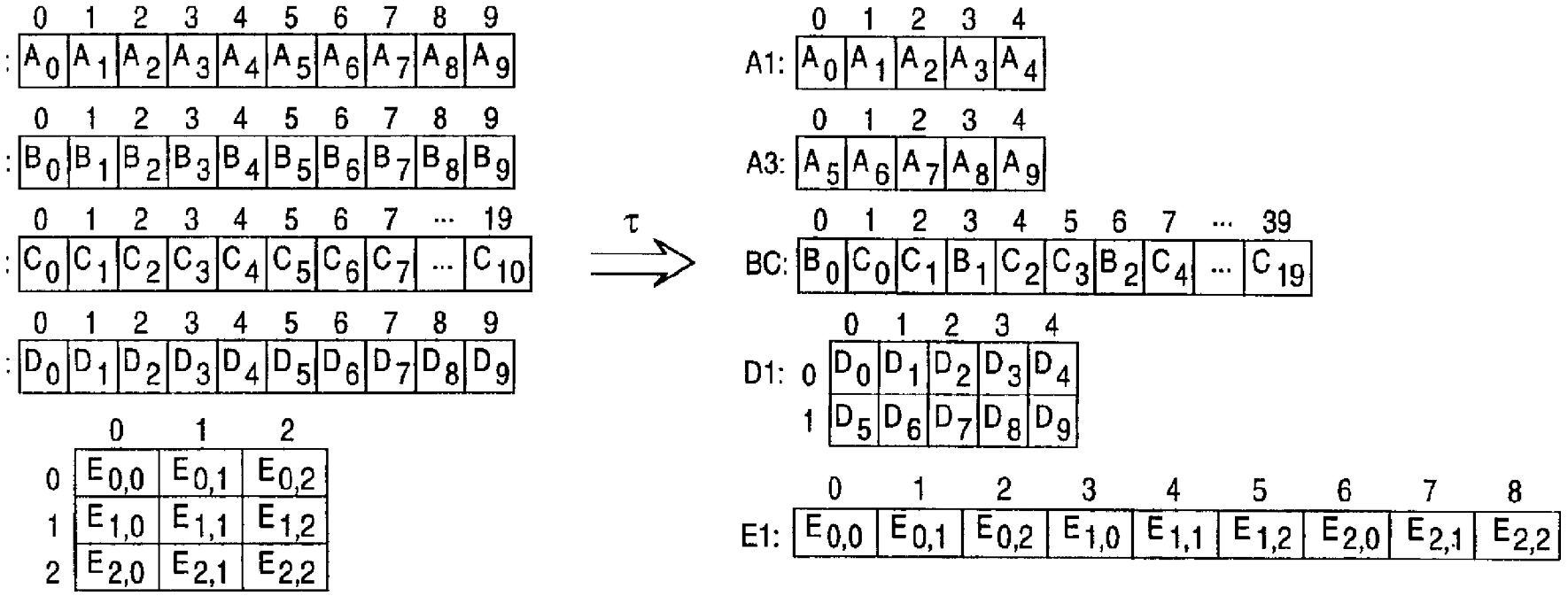

0 1 2 3 4 5 6 7 8 9
: A0 A1 A2 A3 A4 A5 A6 A7 A8 A9
0 1 2 3 4 5 6 7 8 9
: B0 B1 B2 B3 B4 B5 B6 B7 B8 B9
0 1 2 3 4 5 6 7 ... 19
: C0 C1 C2 C3 C4 C5 C6 C7 ... C10
0 1 2 3 4 5 6 7 8 9
: D0 D1 D2 D3 D4 D5 D6 D7 D8 D9
0 1 2
0 E0,0 E0,1 E0,2
1 E1,0 E1,1 E1,2
2 E2,0 E2,1 E2,2
τ
A1: 0 1 2 3 4 A0 A1 A2 A3 A4
A3: 0 1 2 3 4 A5 A6 A7 A8 A9
BC: 0 1 2 3 4 5 6 7 ... 39 B0 C0 C1 B1 C2 C3 B2 C4 ... C19
D1: 0 1 2 3 4 0 D0 D1 D2 D3 D4 1 D5 D6 D7 D8 D9
E1: 0 1 2 3 4 5 6 7 8 E0,0 E0,1 E0,2 E1,0 E1,1 E1,2 E2,0 E2,1 E2,2

**FIG. 24b**

[Drawing 25]

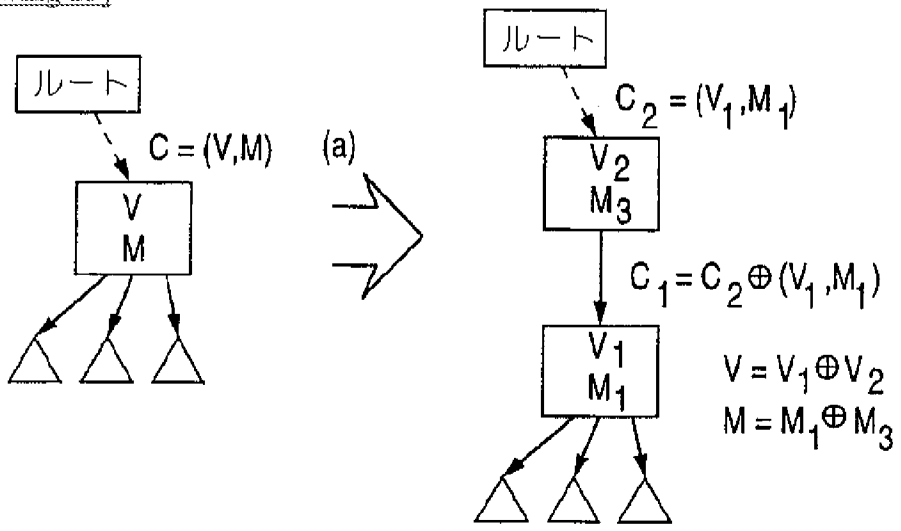



FIG. 25a

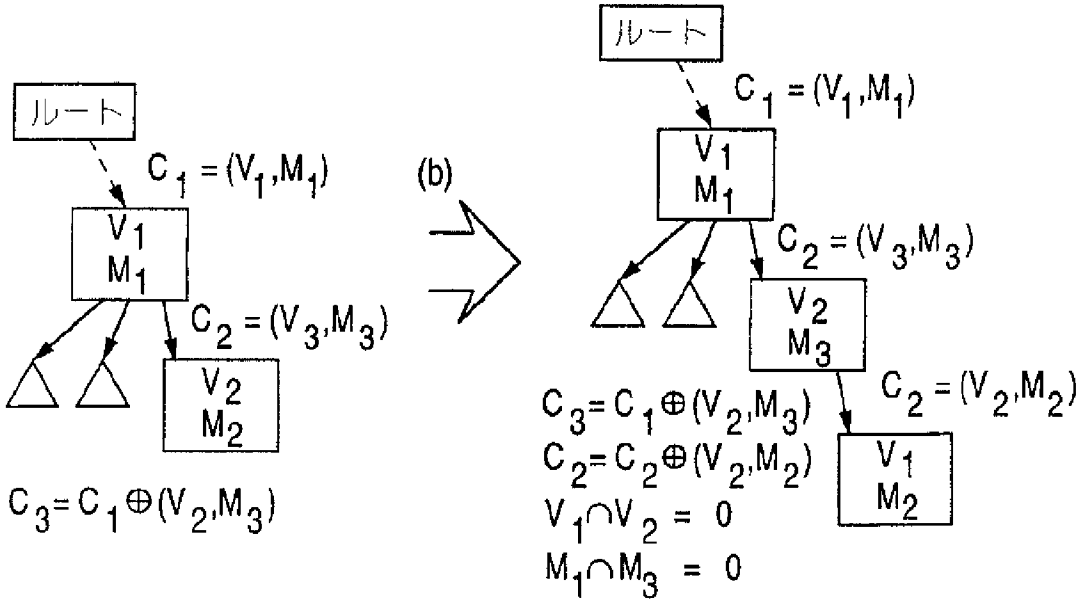



FIG. 25b

[Drawing 25]

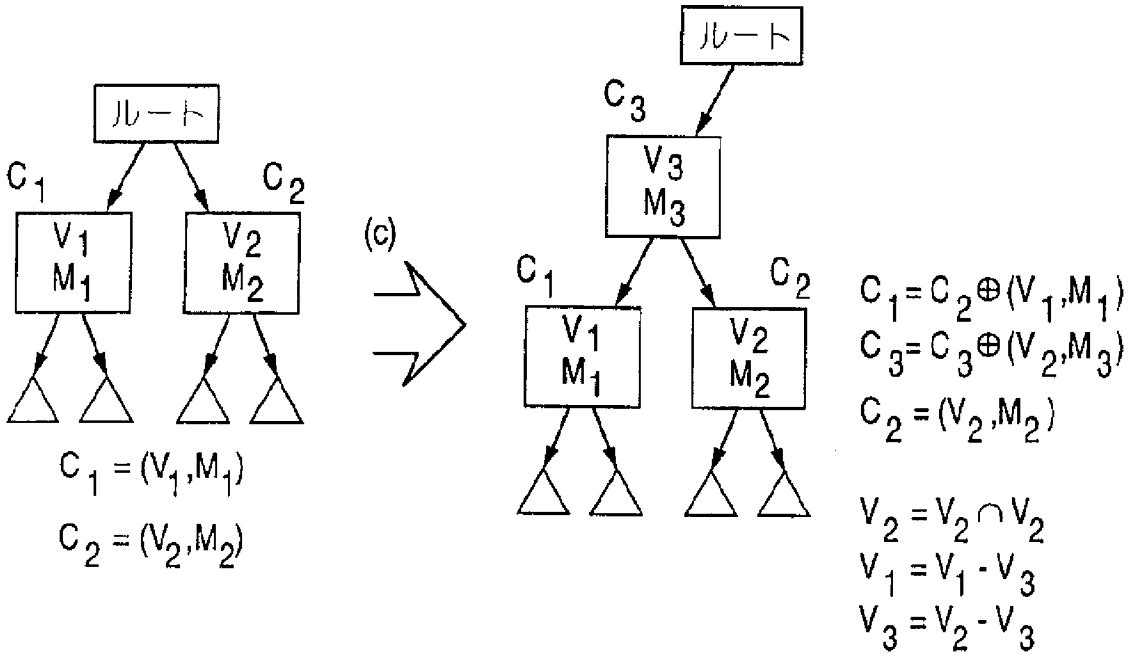



FIG. 25c

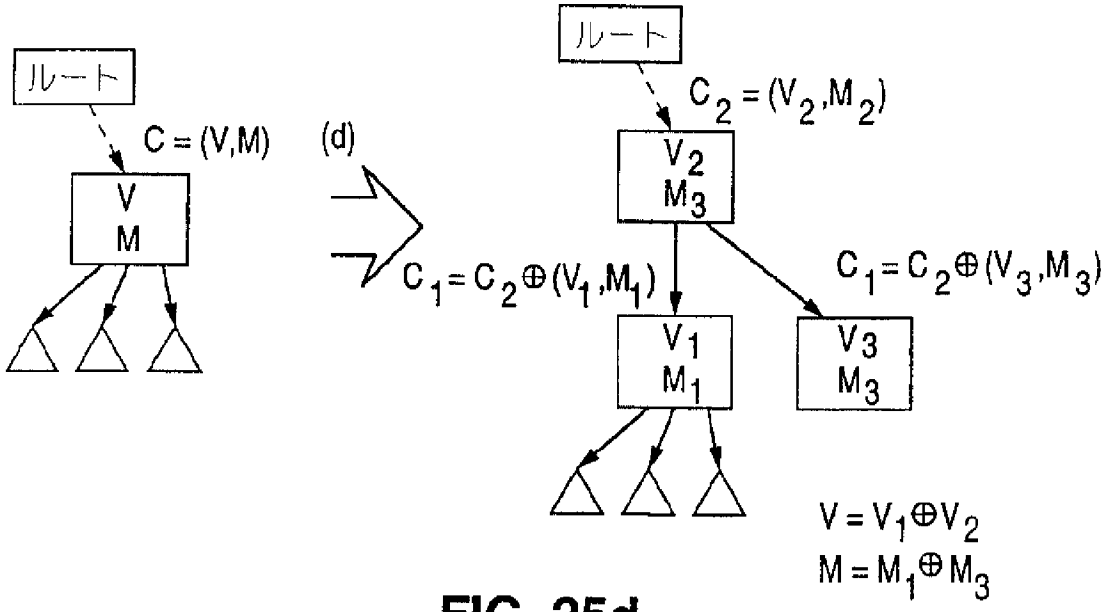
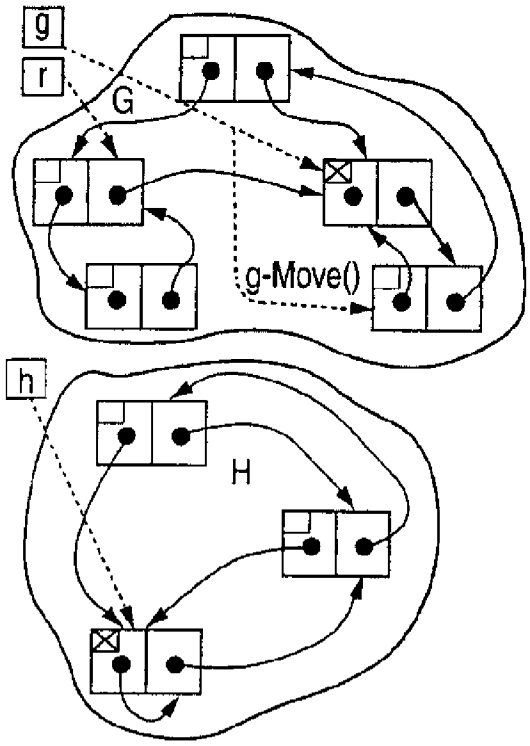



FIG. 25d

[Drawing 26]




```
Node g, h;
method P(...,Node f) {
  /*1*/  g = g.Move();
           h = h.Move();
  /*2*/  h = h.Insert(new Node);
                ⋮
  /*3*/  x.R(...,f.Move());
                ⋮
  /*4*/  if (f==g) ?...

  /*5*/  if (g==h) F...
                ⋮
  /*6*/  f.Token=False;
           g.Token=True;

  /*7*/  if (f.Token)? ...
                ⋮
  /*8*/  f.Token=True;
           h.Token=False;

  /*9*/  if (f.Token) T ...
}
```

FIG. 26

[Drawing 30]

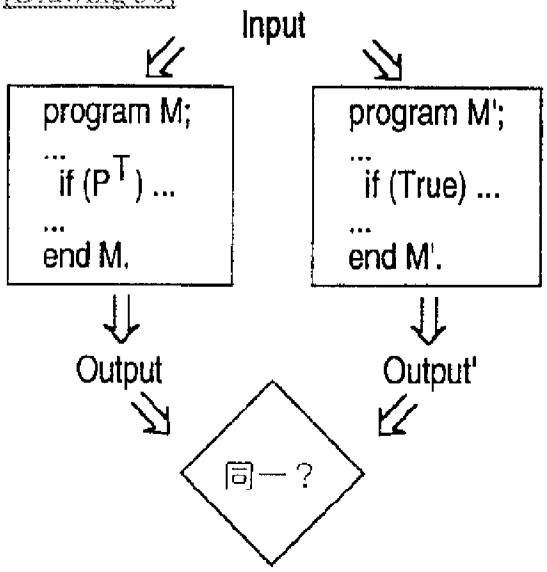



FIG. 30

[Drawing 27]

```
thread S {
  int R;
  while (1) {
    R = random(1,C);
    X = R*R;
    sleep(3);
  }
}
```

```
thread T {
  int R;
  while (i) {
    R = random(1,C);
    X = 7*R*R;
    sleep(2);
    X *= X;
    sleep(5);
  }
}
```

```
int X, Y;
const C = sqrt(maxint)/10;
main () {
  S.run(); T.run();
      ...
  if ((Y - 1) == X) F <= P
      ...
}
```

**FIG. 27**

[Drawing 28]

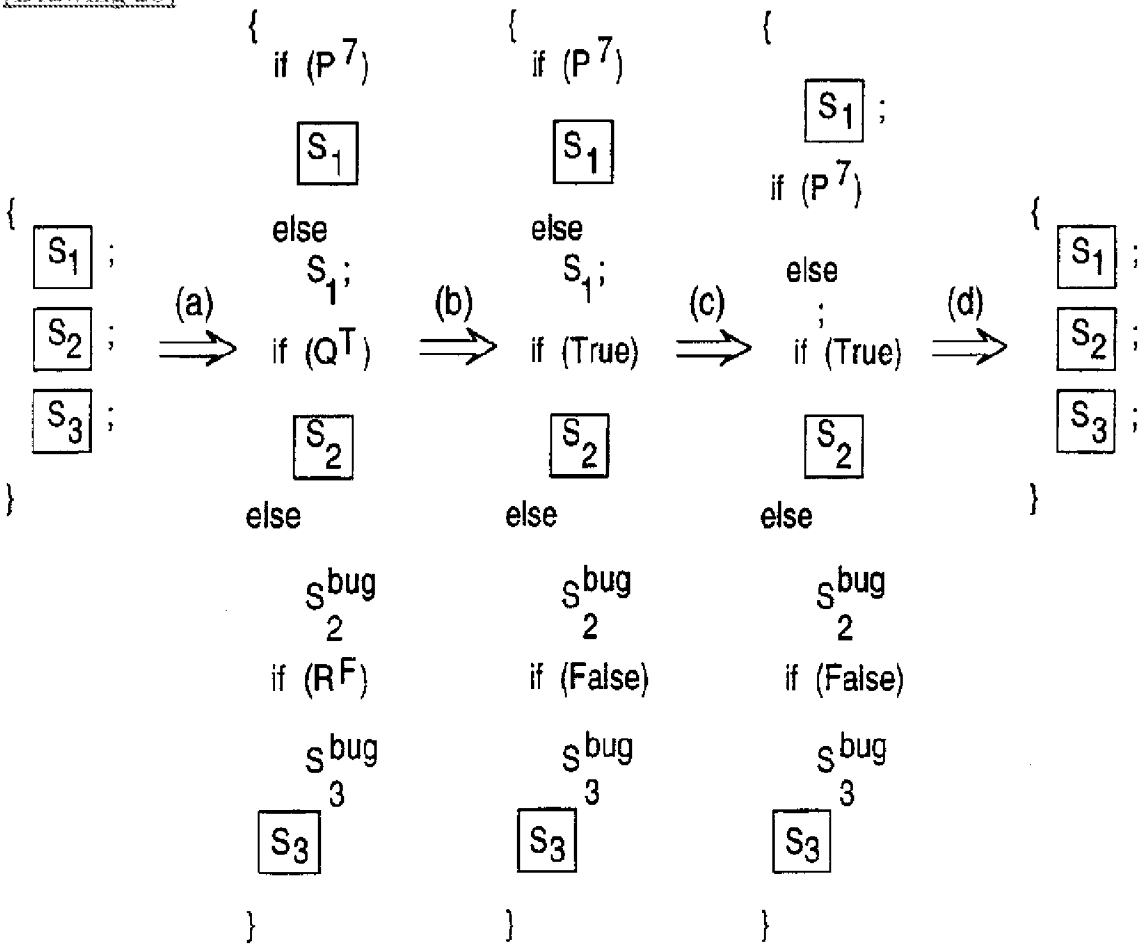



**FIG. 28**

[Drawing 29]

Drawing 311

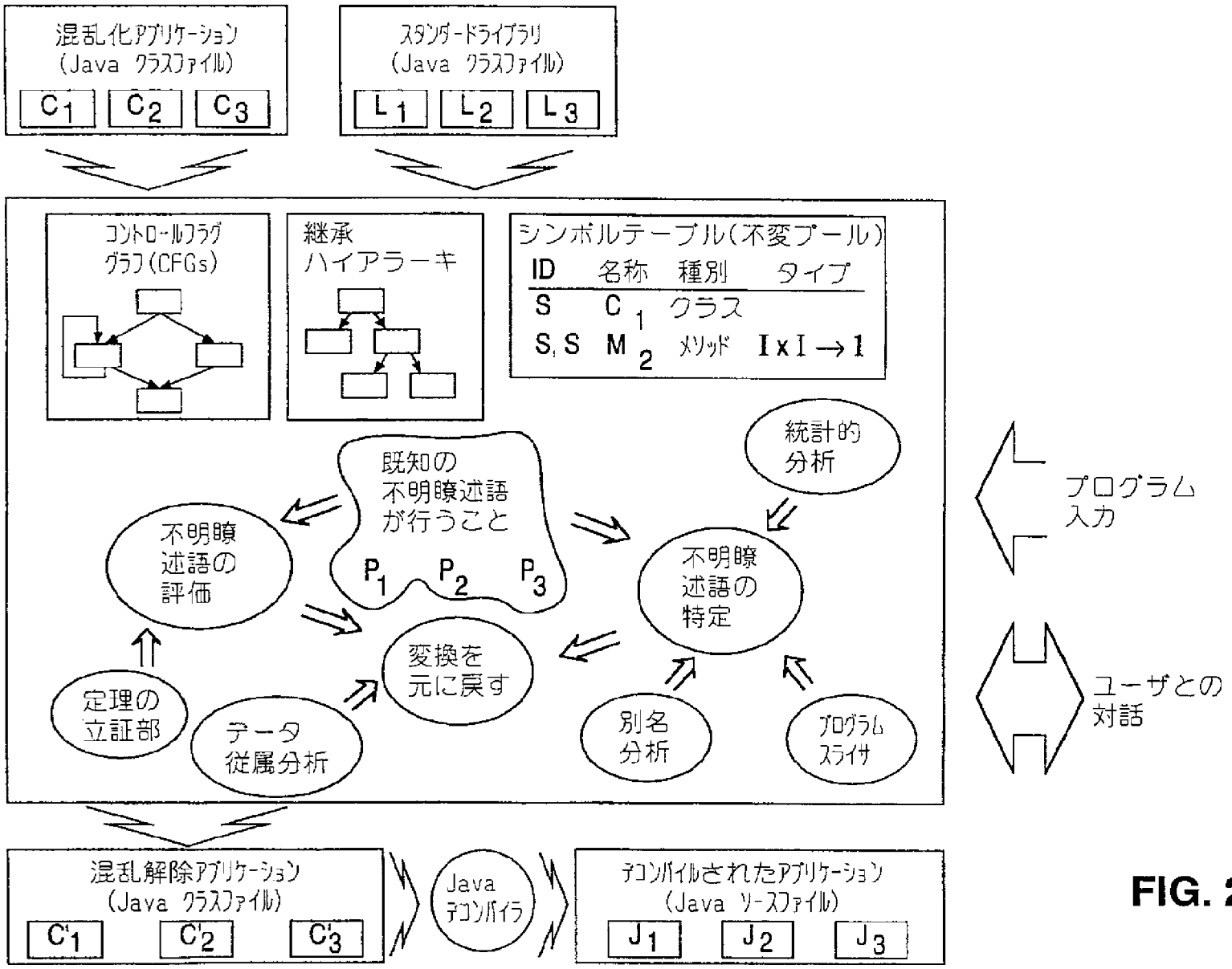



FIG. 29

| 混乱化 目標 | 操作 | 変換 | 品質 効力 | 弾力性 | コスト | メトリクス | セクション |
|---|---|---|---|---|---|---|---|
| レイアウト | | IDのスクランブル | 中 | ワンウェイ | 無料 | | 5.5 |
| | | フォーマッティングの変更 | 低 | ワンウェイ | 無料 | | 5.5 |
| | | コメントの除去 | 高 | ワンウェイ | 無料 | | 5.5 |
| コントロール | 計算 | デッド又はイレリバントコードの挿入 | 不明瞭述語の品質と、コンストラクトが挿入される入れ子深さとに依存する | | | μ1, μ2, μ3 | 6.2.1 |
| | | ループ条件の拡張 | | | | μ1, μ2, μ3 | 6.2.2 |
| | | 可約から非可約へ | | | | μ1, μ2, μ3 | 6.2.3 |
| | | 冗長オペランドの付加 | | | | μ1 | 6.2.6 |
| | | プログラム用イディオムの除去 | 中 | 強 | † | μ1 | 6.2.4 |
| | | テーブル解釈 | 高 | 強 | 高価 | μ1 | 6.2.5 |
| | 集合 | インラインメソッド | 中 | ワンウェイ | 無料 | μ1 | 6.3.1 |
| | | アウトライン文 | ロ | 強 | 無料 | μ1 | 6.3.1 |
| | | インタリーブメソッド | 不明瞭述語の品質に依存する | | | μ1, μ2, μ3 | 6.3.2 |
| | | クローンメソッド | | | | μ1, μ7 | 6.3.3 |
| | | ブロックループ | 低 | 弱 | 無料 | μ1, μ2 | 6.3.4 |
| | | アンロールループ | 低 | 弱 | 安価 | μ1 | 6.3.4 |
| | | ループ分裂 | 低 | 弱 | 無料 | μ1, μ2 | 6.3.4 |
| | オーダリング | 再オーダー文 | 低 | ワンウェイ | 無料 | | 6.4 |
| | | 再オーダーループ | 低 | ワンウェイ | 無料 | | 6.4 |
| | | 再オーダー式 | 低 | ワンウェイ | 無料 | | 6.4 |

FIG. 31a-1

[Drawing 31]

スカラーからオブジェクトへプロンプトする

| 混乱化 目標 | 操作 | 変換 | 品質 効力 | 弾力性 | コスト | メトリクス | セクション |
|---|---|---|---|---|---|---|---|
| データ | 記憶&符号化 | 符号化の変更 | 符号化関数の複雑性に依存する | | | μ1 | 7.1.1 |
| | | | 低 | 強 | 無料 | | 7.1.2 |
| | | 可変寿命の変更 | 低 | 強 | 無料 | μ4 | 7.1.2 |
| | | 変数の分割 | 原変数が分割される変数の数に依存する | | | μ1 | 7.1.3 |
| | | 静的データから手続データへの転換 | 生成関数の複雑性に依存する | | | μ1,μ2 | 7.1.4 |
| | 集合 | 併合スカラー変数 | 低 | 弱 | 無料 | μ1 | 7.2.1 |
| | | ファクタクラス | 中 | † | 無料 | μ1, μ7b,c,e | 7.2.3 |
| | | 挿入Bogus クラス | 中 | † | 無料 | μ1, μ7b,c | 7.2.3 |
| | | 再ファクタクラス | 中 | † | 無料 | μ1, μ7b,c,e | 7.2.3 |
| | | 分割アレイ | † | 弱 | 無料 | μ1, μ2, μ6 | 7.2.2 |
| | | 併合アレイ | † | 弱 | 無料 | μ1, μ3 | 7.2.2 |
| | | フォールドアレイ | † | 弱 | 安価 | μ1,μ2,μ3,μ6 | 7.2.2 |
| | | 平担化アレイ | † | 弱 | 無料 | | 7.2.2 |
| | オーダリング | 再オーダーメソッド&インスタンス変数 | 低 | ワンウェイ | 無料 | | 7.3 |
| | | 再オーダーアレイ | 低 | 弱 | 無料 | | 7.3 |

FIG. 31a-2

[Drawing 31]

| 混乱化 目標 | 操作 | 変換 | 品質 効力 | 弾力性 | コスト | メトリクス | セクション |
|---|---|---|---|---|---|---|---|
| 目標となる | 予防 | HoseMocha | 低 | トリビアル | 無料 | μ1 | 9 |
| | 本来 | スライシングを予防するために別名化形式を付加する | 中 | 強 | 無料 | μ1,μ5 | 9.4 |
| | | スライシングを予防するために変数従属性を付加する | 不明瞭述語の品質に依存する | | | μ1 | 9.4 |
| | | Bogus データ従属性を付加する | 中 | 弱 | 安価 | μ1 | 9.1.1 |
| | | サイド効果を有する不明瞭な述語を使用する | 中 | 弱 | 無料 | μ1 | 9.5 |
| | | ティフィカルト定理を用いて不明瞭な述語を作る | † | † | † | μ1 | 9.5 |

FIG. 31b

[Drawing 32]

| 不明瞭構造 | 品質 | | セクション |
|---|---|---|---|
| | 弾力性 | コスト | |
| ライブラリ関数の呼び出しから生成される | トリビアル | ライブラリ関数のコストに依存する | 6.1.1 |
| ローカル（内部基本ブロック）情報から生成される | トリビアル | 無料...安価 | 6.1.1 |
| グローバル（内部基本ブロック）情報から生成される | 弱 | 無料...安価 | 6.1.1 |
| 内部手続並びに別名情報から生成される | フル | 安価...高価 | 8.1 |
| プロセス相互作用並びにスケジューリングから生成される | フル | 安価...高価 | 8.2 |

FIG. 32

[Translation done.]

* NOTICES *



1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

## CORRECTION OR AMENDMENT

[Kind of official gazette]Printing of amendment by regulation of Patent Law Article 17 of 2
[A section Type] The 3rd Type of the part VI gate
[Publication date]Heisei 18(2006) January 5 (2006.1.5)

[An official announcement number] The ** table 2002-514333 (P2002-514333A)
[An announcement date] Heisei 14(2002) May 14 (2002.5.14)
[Application number]Japanese Patent Application No. 11-508660
[International Patent Classification]
***G06F 21/22 (2006.01***

***G06F 12/00 (2006.01***

*G09C 5/00 (2006.01)*

[FI]
G06F 9/06 660 L

G06F 12/00 537 H

G09C 5/00

[A Written Amendment]
[Filing date]Heisei 17(2005) June 9 (2005.6.9)
[Amendment 1]
[Document to be Amended]Description
[Item(s) to be Amended]The passage of the contents of amendment

[Method of Amendment]Change
[The contents of amendment]

(19)日本国特許庁（ＪＰ）　(12)**公表特許公報**（Ａ）　(11)特許出願公表番号
特表2002－514333
（P2002－514333A）
(43)公表日　平成14年５月14日(2002.5.14)

| (51)Int.Cl.[7] | 識別記号 | ＦＩ | | ﾃｰﾏｺｰﾄﾞ(参考) |
|---|---|---|---|---|
| Ｇ０６Ｆ　1/00 | | Ｇ０６Ｆ | 12/00 | ５３７Ｈ |
| 12/00 | ５３７ | Ｇ０９Ｃ | 5/00 | |
| // Ｇ０９Ｃ　5/00 | | Ｇ０６Ｆ | 9/06 | ６６０Ｌ |

審査請求　未請求　　予備審査請求　有　　（全105頁）

(21)出願番号　特願平11－508660
(86)（22)出願日　平成10年６月９日(1998.6.9)
(85)翻訳文提出日　平成11年12月９日(1999.12.9)
(86)国際出願番号　ＰＣＴ／ＵＳ９８／１２０１７
(87)国際公開番号　ＷＯ９９／０１８１５
(87)国際公開日　平成11年１月14日(1999.1.14)
(31)優先権主張番号　３２８０５７
(32)優先日　平成９年６月９日(1997.6.9)
(33)優先権主張国　ニュー・ジーランド（ＮＺ）

(71)出願人　インタートラスト　テクノロジーズ　コーポレイション
アメリカ合衆国，カリフォルニア　94086，サニーベール，オークミード　パークウエイ　460
(72)発明者　コールベルウ，クリスチャン　スペン
ニュージーランド国，オークランド，マウント　エデン，グレナルモンド　ロード　25
(74)代理人　弁理士　石田　敬　（外４名）

最終頁に続く

(54)【発明の名称】　ソフトウェアセキュリティを増強するための混乱化技術

(57)【要約】
本発明は、ソフトウェアのセキュリティを増強するための混乱化技術を提供する。１つの実施形態においては、ソフトウェアセキュリティを増強するための混乱化技術向けの方法には、混乱化するべきコードのサブセット（例えば１つのアプリケーションのコンパイルされたソースコード）を選択する段階及びコードの選択されたサブセットを混乱化する段階が内含される。混乱化には、コードの選択されたサブセットに対し混乱化変換を適用することが含まれる。変換されたコードは、未変換コードに対する弱い等価性をもつものでありうる。適用される変換は、望まれるセキュリティレベル（例えばリバースエンジニアリングに対する耐性）に基づいて選択されうる。適用される変換には、エイリアシング及び同時実行技術を用いて構築されうる、不明瞭なコンストラクトを用いて新規作成することでありうる制御変換が含まれる可能性がある。従って、望ましい混乱化レベルに基づいて（例えば望ましい効力、弾力性及びコストに基づいて）増強されたソフトウェアセキュリティを得るためにコードを混乱化することが可能である。

【特許請求の範囲】

１．コンピュータが実施するコードを混乱化するための方法であって、

混乱化すべきコードのサブセットを選択する段階；

適用すべき混乱化変換を選択する段階；及び

変換を適用する段階；

を含んで成り、変換されたコードは未変換コードに対する弱い等価性を提供する方法。

２．処理すべきアプリケーションのコードのため、ソースコードに対応する単数又は複数のソースコード入力ファイルを識別する段階；

必要とされる混乱化レベル（効力）を選択する段階；

最大実行時間又は空間ペナルティ（コスト）を選択する段階；

入力ファイルを読取りかつ構文解析する段階；

処理すべきアプリケーションによって使用されるデータタイプ、データ構造、及び制御構造を識別する情報を提供する段階；

必要とされる効力が選択されるか又は最大コストを上回ってしまうまで、ソースコードオブジェクトに対して混乱化変換を選択し適用する段階；及び

アプリケーションの変換済みコードを出力する段階；

をさらに含む、請求項１に記載のコンピュータにより実施される方法。

３．変換には、不明瞭なコンストラクトが含まれ、この不明瞭コンストラクトは、エイリアシング及び同時実行技術を用いて構築されている、請求項１に記載の方法。

４．混乱化されたコードに対し適用された混乱化変換についての

情報及びそのアプリケーションのソースコードに対する変換済みアプリケーションの混乱化されたコードに関する情報を出力する段階をさらに含んで成る、請求項１に記載の方法。

５．変換は、１つのアプリケーションのコードの可観測性挙動を保存するように選択される、請求項１に記載の方法。

６．コードを混乱化解除する段階をさらに含んで成り、コードの混乱化解除に

は、スライシング、部分評価、データフロー分析又は統計分析を使用することにより１つのアプリケーションの混乱化されたコードからあらゆる混乱化を除去する段階が含まれる、請求項１に記載の方法。

７．コードを混乱化するためのコンピュータ読取り可能な媒体上に具体化されたコンピュータプログラムであって、

混乱化すべきコードのサブセットを選択する論理；

適用すべき混乱化変換を選択する論理；及び

変換を適用する論理；

を含み、変換されたコードが、未変換コードに対する弱い等価性を提供する、コンピュータプログラム。

８．処理すべきアプリケーションのコードのため、ソースコードに対応する単数又は複数のソースコード入力ファイルを識別する論理；

必要とされる混乱化レベル（効力）を選択する論理；

最大実行時間又は空間ペナルティ（コスト）を選択する論理；

入力ファイルを読取りかつ構文解析する論理；

処理すべきアプリケーションによって使用されるデータタイプ、データ構造、及び制御構造を識別する情報を提供する論理；

必要とされる効力が達成されるか又は最大コストを上回ってしまうまで、ソースコードオブジェクトに対して混乱化変換を選択し適

用する論理；及び

アプリケーションの変換済みコードを出力する論理；

をさらに含む、請求項７に記載のコンピュータプログラム。

９．変換には、不明瞭なコンストラクトが含まれ、この不明瞭コンストラクトは、エイリアシング及び同時実行技術を用いて構築されている、請求項７に記載のコンピュータプログラム。

10．混乱化されたコードに対し適用された混乱化変換についての情報及びそのアプリケーションのソースコードに対する変換済みアプリケーションの混乱化されたコードに関する情報を出力する論理をさらに含む、請求項７に記載のコンピ

ュータプログラム。

11. 変換は、１つのアプリケーションのコードの可観測性挙動を保存するように選択される、請求項７に記載のコンピュータプログラム。

12. コードを混乱化解除する論理をさらに含んで成り、コードの混乱化解除には、スライシング、部分評価、データフロー分析又は統計分析を使用することにより１つのアプリケーションの混乱化されたコードからあらゆる混乱化を除去する段階が含まれる、請求項７に記載のコンピュータプログラム。

13. コードを混乱化するための装置であって、

混乱化すべきコードのサブセットを選択するための手段；

適用すべき混乱化変換を選択するための手段；及び

変換を適用するための手段；

を含み、変換されたコードが、未変換コードに対する弱い等価性を提供する、装置。

14. 処理すべきアプリケーションのコードのため、ソースコードに対応する単数又は複数のソースコード入力ファイルを識別するための手段；

必要とされる混乱化レベル（効力）を選択するための手段；

最大実行時間又は空間ペナルティ（コスト）を選択するための手段；

入力ファイルを読取りかつ構文解析するための手段；

処理すべきアプリケーションによって使用されるデータタイプ、データ構造、及び制御構造を識別する情報を提供するための手段；

必要とされる効力が達成されるか又は最大コストを上回ってしまうまで、ソースコードオブジェクトに対して混乱化変換を選択し適用するための手段；

アプリケーションの変換済みコードを出力するための手段；

をさらに含んで成る、請求項１３に記載の装置。

15. 変換には、不明瞭なコンストラクトが含まれ、この不明瞭コンストラクトは、エイリアシング及び同時実行技術を用いて構築されている、請求項１３に記載の装置。

16. 混乱化されたコードに対し適用された混乱化変換についての情報及びその

アプリケーションのソースコードに対する変換済みアプリケーションの混乱化されたコードに関する情報を出力するための手段をさらに含んで成る、請求項１３に記載の装置。

17. 変換は、１つのアプリケーションのコードの可観測性挙動を保存するように選択される、請求項１３に記載の装置。

18. コードを混乱化解除するための手段をさらに含んで成り、コードの混乱化解除には、スライシング、部分評価、データフロー分析又は統計分析を使用することにより１つのアプリケーションの混乱化されたコードからあらゆる混乱化を除去する段階が含まれる、請求項１３に記載の装置。

19. コードがJava™バイトコードを含む、請求項１３に記載の装置。

20. 変換が、データ混乱化，制御混乱化又は予防混乱化を提供する、請求項１３に記載の装置。

【発明の詳細な説明】
ソフトウェアセキュリティを増強するための混乱化技術
発明の分野
本発明は、ソフトウェアの解釈、復号又はリバースエンジニアリングを防止又は少なくとも妨害するための方法及び装置に関する。より特定的に言うと、排他的ではないものの、本発明は、ソフトウェアから識別可能な構造又は情報を、逆コンパイル又はリバースエンジニアリングプロセスがさらに困難になるような形で挿入し、除去し又は再配置することによって、ソフトウエアの構造的かつ論理的複雑性を増大させるための方法及び装置に関する。
発明の背景
ソフトウェアは、その性質上第３者により分析されコピーされ易い。これまでにもソフトウェアセキュリティを増強させるために多大な努力が払われてきており、これらの成功はさまざまなものであった。このようなセキュリティの問題は、ソフトウェアの無許可コピーを防止する必要性及びリバースエンジニアリングを介して決定できるようなプログラミング技術を隠したいという願望に関するものである。
例えば著作権の様な確立された法的方法は、立法上の保護措置を提供する。しかしながら、このような制度下で作り出された法的権利を主張することは、費用及び時間が共にかかる作業でありうる。さらに、著作権の下でソフトウエアに対し付与される保護は、プログラミング技術をカバーしていない。かかる技術（すなわち、ソフトウェアの形態に反する機能）は、法律上保護するのが困難である。リバースエンジニアは、問題のソフトウェアの機能の詳細な知識に基づき、最初から関連するソフトウェアを書直すことによって権利侵害から免れることができる。かかる知識は、データ構造、抽象化及びコードの組織を分析することから導き出すことができる。
ソフトウエア特許は、より広範な保護を提供してくれる。しかしながら、ソフトウェアの法的保護を技術的保護と結合させることは明らかに有利である。
所有権主張可能なソフトウェアの保護に対する従来のアプローチは、暗号化に

基づいたハードウェア上の解決法を使用するか又は、ソースコード構造の単純な再配置に基づくもののいずれかであった。ハードウェアに基づく技術は、それが一般に費用のかかるものであり、特定のプラットフォーム又はハードウェアアドオンに結びつけられるものであるという点で、理想的ではない。ソフトウェアによる解決法は、標準的に、Java™用のCrema混乱化器といったトリビアルコード混乱化器を内含する。一部の混乱化器は、アプリケーションの語い構造を目標とし、通常ソースコードフォーマッティング及びコメントを除去し、変数を再命名する。しかしながら、このような混乱化技術は、悪意あるリバースエンジニアリングに対する充分な保護を提供しない。すなわち、リバースエンジニアリングは、ソフトウェアが分散される形とは無関係の問題である。さらに、ソフトウェアが、オリジナルソースコード内の情報の多く又は全てを保持するハードウェア依存性フォーマットで分散されている場合、問題はさらに悪化する。かかるフォーマットの例としては、Java™バイトコード及びアーキテクチュアニュートラル分散フォーマット（ＡＮＤＦ）がある。

ソフトウェア開発には、プログラマが多大な時間、努力そして技能を投入している可能性がある。商業的には、所有権主張可能な技術を競合者がコピーするのを防止できることは、きわめて重要である。

発明の開示

本発明は、リバースエンジニアリングに対するソフトウェアの抵抗力を高めるため（又は一般大衆に対し有用な選択肢を提供するため）のコンピュータにより実施される方法といったような、ソフトウェアセキュリティ用の混乱化技術のための方法及び装置を提供する。一実施形態においては、コードを混乱化するためにコンピュータにより実施される方法には、１個またはそれ以上のコードに対する混乱化変換の供給の完了についてテストする段階、混乱化すべきコードのサブセットを選択する段階、適用すべき混乱化変換を選択する段階、変換を適用する段階そして完了テスト段階へと復帰する段階が内含される。

１つの変形実施形態においては、本発明は、コンピュータ上で実行され、記憶され、又はそれにより操作されるソフトウェアが、予め定められ制御された程度

のリバースエンジニアリング耐性を示すような形でコンピュータを制御する方法において、ソフトウェアの選択された部分に選択された混乱化変換を適用する段階であって、必要とされる程度のリバースエンジニアリング耐性、ソフトウェアのオペレーションにおける有効性及び変換されたソフトウェアのサイズを提供するように選択された混乱化を用いて一定のレベルの混乱化を達成する段階；及び混乱化変換を反映するべくソフトウェアを更新する段階を内含する方法に関する。

好ましい実施形態においては、本発明は、ソフトウェアのセキュリティを増強するためのコンピュータにより実施される方法において、処理すべきアプリケーションのためソースソフトウェアに対応

する１個またはそれ以上のソースコード入力ファイルを識別する段階；必要とされる混乱化レベル（例えば効力）を選択する段階；最大実行時間又は空間ペナルティ（例えばコスト）を選択する段階；任意にはソースコードにより直接又は間接的に読取られた任意のライブラリ又は補足的ファイルと共に入力ファイルを読取りかつ構文解析する段階；処理すべきアプリケーションによって使用されるデータタイプ、データ構造、及び制御構造を識別する情報を提供し、この情報を記憶するべく適切なテーブルを構築する段階、前処理段階に応えてアプリケーションについての情報を前処理する段階、ソースコードオブジェクトに対して混乱化コード変換を選択し適用する段階；必要とされる効力が達成されるか又は最大コストを上回ってしまうまで混乱化コード変換段階を反復する段階、及び変換済みソフトウェアを出力する段階を含んで成る方法を提供する。

好ましくは、アプリケーションに関する情報は、さまざまな静的分析技術及び動的分析技術を用いて得られる。静的分析技術としては、手順間データフロー分析及びデータ従属性分析が含まれる。動的分析技術としては、プロファイリングが含まれ、任意には、ユーザを介して情報を得ることができる。プロファイリングは、特定のソースコードオブジェクトに対し適用することが可能な混乱化レベルを決定するのに使用することができる。変換は、複数の不明瞭コンストラクト（構文）を使用して作成された制御変換を含むことができる。この不明瞭コンス

トラクトは、性能の見地から見て実行するのに廉価で混乱化器が簡単に構築できしかも混乱解除器がそれを破断するのには高価である、何れかの数学的オブジェクトである。好ましくは、不明瞭コンストラクトは、別名化及び同時実行技術を用いて構築されうる。ソースアプリケーションに関する情報は同様に、そのアプリケーションが含有するプログラム用イディオム及び

言語コンストラクトの性質を決定するプラグマティック（実用）分析を用いて得ることができる。

混乱化変換の効力は、ソフトウェア複雑性メトリックを用いて評価可能である。混乱化コード変換は、あらゆる言語コンストラクトに応用できる。例えば、モジュール、クラス又はサブルーチンを分割又は併合することができ；新しい制御及びデータ構造を作成することもでき；又、オリジナル制御及びデータ構造を修正することもできる。好ましくは、変換されたアプリケーションに対し付加される新しいコンストラクトは、前処理の間に収集したプラグマティック情報に基づいて、ソースアプリケーション内のものにできるかぎり類似したものとなるように選択される。このメソッドは、混乱化変換がそれについて適用された情報及びソースソフトウェアに対する変換済みアプリケーションの混乱化されたコードに関する情報を含む補助ファイルを生成することができる。

好ましくは、混乱化変換は、Ｐが未変換ソフトウェアであり、Ｐ’が変換済みソフトウェアである場合、ＰとＰ’が同じ可観測性挙動を有するような形で、ソフトウェアの可観測性挙動を保つように選択される。より特定的に言うと、Ｐが終結できないか又はエラー条件を伴って終結した場合、Ｐ’は終結してもしなくてもよく、そうでなければＰ’は終結してＰと同じ出力を生成する。可観測性挙動としては、ユーザが経験する効果が含まれるが、Ｐ及びＰ’は、ユーザにとって観測不能な異なる詳細な挙動を伴って走行することができる。例えば、異なるものでありうるＰ及びＰ’の詳細な挙動としては、ファイル作成、メモリ使用及びネットワーク通信が含まれる。

一実施形態においては、本発明は同様に、スライシング、部分評価、データフロー分析又は統計分析を用いることにより、混乱化済

みアプリケーションから混乱化を除去するために採用される混乱解除用ツールを提供する。

図面の簡単な説明

以下に、単なる一例として、図面を参照して本発明を説明する。

図１は、本発明の教示に従ったデータ処理システムを示す。

図２は、混乱化変換のカテゴリを含むソフトウェア保護の分類を例示する。

図３ａ及び３ｂは、（ａ）サーバー側実行及び（ｂ）部分的サーバ側実行によりソフトウエアセキュリティを提供するための技術を示す。

図４ａ及び４ｂは、（ａ）暗号化及び（ｂ）署名されたネーティブコードを用いることによりソフトウエアセキュリティを提供するための技術を示す。

図５は、混乱化を通してソフトウエアセキュリティを提供するための技術を示す。

図６は、Java™アプリケーションと共に使用するのに適した混乱化ツールの一例のアーキテクチャを例示する。

図７は、既知のソフトウエア複雑性メトリックセレクションを作表したテーブルである。

図８ａ及び８ｂは、混乱化変換の弾力性を例示している。

図９は、異なるタイプの不明瞭述語を示す。

図１０ａ及び１０ｂは、（ａ）トリビアルな不明瞭コンストラクト及び（ｂ）弱い不明瞭コンストラクトの例を提供している。

図１１は、計算変換（分岐挿入変換）の一例を示す。

図１２ａ～１２ｄは、ループ条件挿入変換を例示している。

図１３は、可約フローグラフを非可約フローグラフに変換する変換を例示する。

図１４は、データ従属性を全く含まない場合に、コード区分を並列化できることを示している。

図１５は、データ従属性を全く含まないコード区分を、適切な同期化基本命令を挿入することによって並行スレッドへと分割できることを示している。

図１６は、いかにして手順Ｐ及びＱがその呼出しサイトでインラインにされ、次にコードから除去されるかを示す。

図１７は、インライン処理メソッド呼出しを例示している。

図１８は、同じクラス内で宣言された２つのメソッドをインタリーブするための技術を示す。

図１９は、オリジナルコードに対して異なる混乱化変換セットを適用することにより１つのメソッドの複数の異なるバージョンを作成するための技術を示す。

図２０ａ～２０ｃは、（ａ）ループブロック化、（ｂ）ループアンローリング及び（ｃ）ループ分裂を含むロール変換の例を提供している。

図２１は、可変的分割例を示す。

図２２は、ストリング「ＡＡＡ」，「ＢＡＡＡＡ」及び「ＣＣＢ」を混乱化するように構築された１つの関数を提供している。

図２３は、２つの３２ビット変数ｘ及びｙを１つの６４ビット変数ｚへと併合する例を示す。

図２４は、アレイ再構成のためのデータ変換の例を示す。

図２５は、継承階層の修正を例示する。

図２６は、オブジェクト及び別名から構築された不明瞭述語を例示する。

図２７は、スレッドを用いた不明瞭コンストラクトの一例を提供

している。

図２８ａ～図２８ｄは、（ａ）が、混乱化中の３つの文$S_{1-3}$を含むオリジナルプログラムを示し、（ｂ）が「恒常な」不明瞭述語を識別する混乱解除器を示し、（ｃ）が文の中の共通コードを決定する混乱解除器を示し、（ｄ）が、いくつかの最終的単純化を適用しプログラムをそのオリジナル形態に戻す混乱解除器を示している、混乱化対混乱解除の関係を例示する図である。

図２９は、Java™混乱解除ツールのアーキテクチャを示す。

図３０は、評価に使用される統計分析例を示す。

図３１ａ及び３１ｂは、さまざまな混乱化変換の概要のテーブルを提供する。

図３２は、さまざまな不明瞭コンストラクトの概要を提供する。

発明の詳細な説明

以下の記述は、出願人が現在開発中であるJava™混乱化ツールに関連して提供されるものである。ただし、当業者であれば、当該技術がその他のプログラミング言語にも適用可能であることは明白であり、本発明はJava™アプリケーションに制限されるものとみなされるべきではない。本発明のその他のプログラミング言語に関連しての実施は、当業者の視野内に入るものとみなされる。以下の実施例は、明確さを期して、特定的に、Java™混乱化ツールを目標としている。

以下の記述においては、次のような名称が用いられることになる。すなわち、Ｐは混乱化されるべき入力アプリケーションである；Ｐ’は変換済みアプリケーションである；Ｔは、それがＰをＰ’へと変換するような形での変換である。Ｐ（Ｔ）Ｐ’は、Ｐ及びＰ’が同じ可観測性挙動を有する場合の混乱化変換である。可観測性挙動は一般に、ユーザが経験する挙動として定義される。かくして、Ｐ’は、ユーザがそれを体験しないかぎりにおいて、Ｐがもたないファイルを作成するといったような予期せぬ効果を有する可能性がある。ＰとＰ’は必ずしも同等の効率をもつ必要はない。

ハードウェア例

図１は、本発明の教示に従ったデータ処理システムを例示する。図１は、３つの主要な要素を含むコンピュータ１００を示す。コンピュータ１００には、このコンピュータのその他の部分へ及びこの部分から適切に構造化された形態で情報を通信するのに使用される入出力（Ｉ／Ｏ）回路１２０が内含されている。コンピュータ１００には、Ｉ／Ｏ回路１２０及びメモリ１４０（例えば揮発性及び不揮発性メモリ）と通信状態にある制御処理ユニット（ＣＰＵ）１３０が内含されている。これらの要素は、大部分の汎用コンピュータに標準的に見られるものであり、実際、コンピュータ１００は、広範なカテゴリのデータ処理デバイスを代表するものとなるよう意図されている。ラスター表示モニター１６０がＩ／Ｏ回路１２０と通信状態で示され、ＣＰＵ１３０が生成する画素を表示するように命じられている。任意の周知のさまざまな陰極線管（ＣＲＴ）又はその他のタイプ

の表示装置を、表示装置１６０として使用することができる。従来のキーボード１５０も、Ｉ／０１２０と通信状態で示されている。当業者であれば、コンピュータ１００が、より大きいシステムの一部でありうることが理解される。例えば、コンピュータ１００は、１つのネットワーク（例えばローカルエリアネットワーク（ＬＡＮ）に接続されたもの）と通信状態にあってもよい。

特に、コンピュータ１００は、本発明の教示に従ってソフトウエアセキュリティを増強するための混乱化回路を内含することができ

、或いは又、当業者であればわかるように、本発明をコンピュータ１００により実行されるソフトウェアの形で実施することも可能である（例えはこのソフトウェアをメモリ１４０内に格納してＣＰＵ１３０上で実行することができる）。例えば、メモリ１４０内に格納された未混乱化プログラムＰ（例えばアプリケーション）を、本発明の１実施形態に従ってメモリ１４０内に記憶された混乱化済みプログラムＰ’を提供するべくＣＰＵ１３０上で実行する混乱化器により、混乱化させることが可能である。

詳細な記述の概要

図６は、Java™混乱化器のアーキテクチャを示す。本発明の方法に従うと、Java™アプリケーションクラスファイルは、任意のライブラリファイルと共にバスされるc.継承ツリーがシンボルテーブルと共に構築され、全てのシンボルについてのタイプ情報及び全てのメソッドについての制御フローグラフを提供する。ユーザは、任意には、Java™プロファイリングツールによって生成されるように、プロファイリングデータファイルを提供することができる。この情報は、アプリケーションのうちの頻繁に実行される部分が非常に高価な変換によって混乱化されていないことを保証するべく混乱化器を案内するのに使用できる。手順間データフロー分析及びデータ従属性分析といったような標準コンパイラ技術を用いてアプリケーションについての情報が収集される。その中には、ユーザにより提供されるものもあれば、専門的技術によって提供されるものもある。この情報は、適切なコード変換を選択し適用するために使用される。

適切な変換が選択される。大部分の適切な変換を選択する上で使用される支配

的な基準は、選択された変換がコードの残りの部分と

自然に混ざり合うという必要条件を内含している。これは、高い適切性値をもつ変換を奨励することによって対処できる。もう１つの必要条件は、低い実行時間ペナルティで高レベルの混乱化を生み出す変換を奨励すべきであるというものである。後者の点は、効力及び弾力性を最大にしコストを最小限にする変換を選択することによって達成される。

混乱化の優先性が、ソースコードオブジェクトに割当てられる。これは、ソースコードオブジェクトの内容を混乱化することがいかに重要であるかを反映することになる。例えば、特定のソースコードオブジェクトが非常に感応性の高い所有権主張できる材料を含む場合、混乱化優先性が高くなる。各メソッドについて、実行時間ランクが決定され、これは、そのメソッドを実行するのに他のどれよりも多くの時間が費された場合１に等しい。

このとき、アプリケーションは、適切な内部データ構造、適切な変換への各ソースコードオブジェクトからのマッピング、混乱化優先性及び実行時間ランクを打ち立てることによって、混乱化される。混乱化変換は、必要とされる混乱化が達成されるか又は最大実行時間ペナルティを上回ってしまうまで適用される。変換済みアプリケーションは、この時点で書き込まれる。

混乱化ツールの出力は、機能的にオリジナルと等価である新しいアプリケーションである。このツールは、変換がそれについて適用された情報及び混乱化されたコードがいかにオリジナルアプリケーションと関連するかの情報が注釈として付いたJava™ソースファイルをも生成することができる。

ここで、ひきつづきJava™混乱化器に関連して、いくつかの混乱化変換例について記述する。

混乱化変換は、その質に従って評価し分類することができる。変

換の質は、その効力、弾力性及びコストに従って表現できる。変換の効力は、Ｐ’がＰとの関係においていかにあいまいであるかに関係する。このようなメトリックは全て、必然的に人間の認識能力によって左右されることから、比較的不確

かなものとなる。当該目的のためには、その変換の有用性の一尺度として変換の効力を考慮するだけで充分である。変換の弾力性は、変換が自動混乱解除器からの攻撃に対しいかにうまく持ちこたえるかを測定する。これは、プログラマの努力と混乱解除器の努力という２つの因子の組合せである。弾力性は、トリビアルからワンウェイまでの目盛上で測定できる。ワンウェイ変換は、それを逆転させることができないという点で極端なものである。第３の構成要素は、変換実行コストである。これは、変換済みアプリケーションＰ’を使用した結果としてこうむった実行時間又は空間ペナルティである。変換評価のさらなる詳細については、以下の好ましい実施形態の詳細な説明の部分で論述される。混乱化変換の主要な分類は、図２Ｃに示され、詳細は図２ｅ～２ｇに与えられている。

混乱化変換の例は、以下の通りである：混乱化変換は、制御混乱化、データ混乱化、レイアウト混乱化及び予防混乱化にカテゴリー分類されうる。これらのいくつかの例について以下で論述する。

制御混乱化には、集合変換、オーダリング変換及び計算変換が含まれる。

計算変換には、不適切な非機能的文の後ろに実の制御フローを隠すこと，対応する高レベル言語コンストラクトが全く存在しないオブジェクトコードレベルでコードシーケンスを導入すること；及び実の制御フローの抽象化を除去するか又はスプリアスなものを導入することが含まれる。

第１の分類（制御フロー）を考慮すると、サイクロマティック及びネスティング複雑性メトリックは、一片のコードの感知された複雑性とそれが含む述語の数の間には強い相関関係があること示唆している。不明瞭述語は、プログラム内に新しい述語を導入する変換の構築を可能にする。

図１１ａを参照すると、不明瞭述語$P^T$が、$S=S_1 \cdots S_n$である基本ブロックＳ内に挿入されている。こうしてＳは半分に分割される。$P^T$述語は、つねに「真」まで評価することになるため、不適切なコードである。図１１ｂでは、Ｓは再び２つの半分に分割され、これらの半分は２つの異なる混乱化済みバージョン$S^a$及び$S^b$へと変換される。従ってリバースエンジニアにとって、$S^a$と$S^b$が同じ機能を果たすことは明白ではなくなる。図１１ｃは、図１１ｂと類似している

が、$S^b$内にバグが導入されている。$P^T$述語はつねに、コード$S^a$の正しいバージョンを選択する。

もう１つのタイプの混乱化変換は、データ変換である。データ変換の一例は、コードの複雑性を増大させるためにアレイを逆構造解釈することである。１つのアレイは、複数のサブアレイに分割でき、２つ以上のアレイは単一のアレイに併合でき、或いは又アレイのディメンションを増大（平坦化）又は減少（フォールディング）させることもできる。図２４は、一定数のアレイ変換例を示している。文（１－２）では、アレイＡが２つのサブアレイＡ１及びＡ２内に分割されている。Ａ１は、偶数の指標をもつ要素を含み、Ａ２は奇数の指標をもつ要素を含む。文（３－４）は、２つの整数アレイＢ及びＣが、１つのアレイＢＣを生み出すべくいかにインタリーブされうるかを例示している。Ｂ及びＣからの要素は、変換済みアレイ全体にわたり均等に拡散される。文（６－７）は、アレイＤのアレイＤ１へのフォールディングを例示している。かかる変換は、従来欠如していたデータ構造を導入するか又は既存のデータ構造を除

去する。こうして、例えば、２次元アレイを宣言する上でプログラマは通常、選ばれた構造が対応するデータ上にマッピングする状態で、１つの目的のためにそれを行なうことから、プログラムのあいまいさが大幅に増大する可能性がある。そのアレイが１－ｄ構造にフォールドされたならば、リバースエンジニアは、貴重なプラグマティック情報を奪われることになるだろう。

もう１つの混乱化変換例は、予防変換である。制御又はデータ変換とは対照的に、予防変換の主たる最終目的は、人間の読み手にとってプログラムをあいまいにすることではなく、既知の自動的混乱解除技術をよりむずかしくするか又は現行の混乱解除器又はデコンパイラ内で既知の問題を開発利用することにある。このような変換は、それぞれ固有の及び目標の変換として知られている。固有予防変換の一例は、for-loopをrun backwardに再オーダすることである。このような再オーダリングは、ループが、ループ支持型データ従属性を全くもたない場合に可能である。混乱解除は同じ分析を行ないループを順方向実行に再オーダすることができる。しかしながら、逆転されたループに、偽りのデータ従属性が付加さ

れた場合、ループの識別及びその再オーダリングは防止されることになる。

混乱化変換のさらなる特定的例について、以下の好ましい実施形態の詳細な説明の部分で論述する。

好ましい実施形態の詳細な説明

オリジナルソースコード内に存在する情報の大部分又は全てを保持する形態でソフトウェアを分散させることが増々一般的になってきている。１つの重要な例が、Javaバイトコードである。かかるコードはデコンバイルが容易であるため、悪意あるリバースエンジニアリングの攻撃の危険性を増大させる。

従って、本発明の１実施形態に従って、ソフトウエアセキュリティの技術的保護のための複数の技術が提供されている。好ましい実施形態の詳細な説明において我々は、自動コード混乱化が、リバースエンジニアリングを防止するための現在最も実現性ある方法であるということを立証して行くつもりである。次に我々は、プログラムを、理解及びリバースエンジニアリングすることがさらに困難な等価物へと変換する混乱化ツールであるコード混乱化器の設計について記述する。

混乱化器は、数多くの場合においてコンバイラ最適化プログラムが使用するものに類似しているコード変換の適用に基づいている。数多くのかかる変換について記述し、それらを分類し、その効力（例えばどの程度まで人間の読み手が当惑させられるか）、弾力性（例えば自動的混乱解除の攻撃にどれほど耐えられるか）及びコスト（例えば、そのアプリケーションに対し、どれほどの性能オーバヘッドが付加されるか）に関しそれらを評価する。

最後に、さまざまな混乱解除技術（例えばプログラムスプライシング）及びそれらに対し混乱化器が利用できると考えられる対策について記述する。

１．序論

充分な時間、努力及び決意が与えられるのであれば、有能なプログラマはつねにどんなアプリケーションにでもリバースエンジニアリングすることができるだろう。アプリケーションに対する物理的アクセスを獲得したリバースエンジニアは、（逆アセンブラ又はデコンバイラを用いて）それをデコンバイルし、次にそ

のデータ構造及び制御フローを分析することができる。これは手動でもでき、又プログラムスライサといったようなリバースエンジニアリングツー

ルを用いて行なうこともできる。

リバースエンジニアリングは新しい問題ではない。しかしながら、大部分のプログラムが大型でモノリシックでかつストリップの状態で出荷されるネーティブコードであったことから、（決して不可能ではないものの）リバースエンジニアリングは難しく、近年になるまで、ソフトウェアデベロッパはリバースエンジニアリングにさほど注意を払ってこなかった、と言う問題がある。

しかしながらこの状況は、デコンパイル及びリバースエンジニアリングをするのが容易な形態でソフトウェアを配布することが増々一般的になるにつれて、変化している。重要な例としては、Javaバイトコード及びアーキテクチャニュートラル分散フォーマット（ＡＮＤＦ）がある。Javaアプリケーションは特に、ソフトウエアデベロッパにとっての問題を提起している。これらは、オリジナルJavaソース情報を事実上全て保持するハードウェア独立の仮想計算機コードである、Javaクラスファイルとして、インタネット上で配布される。従って、これらのクラスファイルはデコンパイルが容易である。しかも、計算の多くが標準ライブラリ内で行なわれることから、Javaプログラムは往々にしてサイズが小さく、従って比較的リバースエンジニアリングしやすい。

Javaデベロッパの主たる関心事は、アプリケーション全ての徹底的な再エンジニアリングではない。このような行動は、明らかに著作権法〔２９〕に違反し訴訟で争うことができることから、比較的価値のないことである。むしろ、デベロッパが最も恐れているのは、競合相手が、そのアプリケーションから所有権主張可能なアルゴリズム及びデータ構造を抽出してそれを自社のプログラム内に取込むことができるようになる、という予想である。これは競合相手に商業的な有効性（開発時間及びコストを削減することによる）を与

えるばかりでなく、検出及び法的追求が困難なことでもある。この最後の点は、法律に関する無限の予算をもつ強力な企業〔２２〕に対して、長期間にわたる法

律上の戦闘を行なう経済力をもたないであろう小規模デベロッパに、特にあてはまることである。

ソフトウェアの法的保護又はセキュリティを提供するためのさまざまな形態の保護の概要が図２に提供されている。図２は、（ａ）悪意あるリバースエンジニアリングに対する保護の種類、（ｂ）混乱化変換の質、（ｃ）混乱化変換により目標とされている情報、（ｄ）レイアウト混乱、（ｅ）データ混乱、（ｆ）制御混乱、及び（ｇ）予防混乱の分類を提供している。

ソフトウエアデベロッパが利用可能である知的財産の技術的保護のさまざまな形態について以下で議論する。この議論はJavaクラスファイルとしてインタネット上で配布されるJavaプログラムに限られるものであるが、結果の大部分は、当業者には明らかであるように、その他の言語及びアーキテクチャニュートラルフォーマットにもあてはまるものである。移動コードの保護に対する唯一の合理的アプローチは、コード混乱化であることを立証していく。さらに我々は、いくつかの混乱化変換を提示し、これらを有効性及び効率に応じて分類し、いかにしてそれらを自動的混乱化ツール内で使用することができるようにするかを示す。

好ましい実施形態の詳細な説明の残りの部分は、以下のように構成される。第２節で、ソフトウェアの盗難に対するさまざまな形態の技術的保護の概要を示し、コード混乱化が現在最も経済的な予防を提供していることを立証する。第３節では、現在構築中であるJavaのためのコード混乱化器であるKavaの設計の簡単な概要を示す。第４節及び５節は、異なるタイプの混乱化変換を分類し評価するのに使用される基準について記述している。第６節、７節、８節及び９節は、混乱化変換のカタログを提示している。第１０節で、より詳細な混乱化アルゴリズムを示している。第１１節では、結果のまとめ及びコード混乱化の将来の方向についての議論で締めくくっている。

２．知的財産の保護

以下の筋書きを考える。アリスは、インタネット上で彼女のアプリケーションをユーザが恐らくは有料で利用できるようにすることを望んでいる、小規模なソフトウエアデベロッパである。ボブは、自分がアリスのアプリケーションのキー

アルゴリズム及びデータ構造にアクセスできた場合に、アリスに対して商業的な優位性を得ることができると考えているライバルのデベロッパである。

これは、２人の敵対者、すなわち自らのコードを攻撃から保護しようとするソフトウエアデベロッパ（アリス）及び、アプリケーションを分析しそれを容易に読取り理解できる形態に変換することを仕事とするリバースエンジニア（ボブ）、の間の２プレイヤーゲームと考えることができる。ここで、ボブがアプリケーションをアリスのオリジナルソースに幾分か近いものに変換することは不要である、という点に留意されたい。必要なのは、リバースエンジニアリングされたコードがボブ及び彼のプログラマにとって理解可能なものである、ということだけである。同様に、アリスはボブから自らのアプリケーション全体を保護する必要すらない可能性がある、ということにも留意されたい。これは恐らく、大部分が、競合相手にとって実際関心の的でない「バタ付きパンコード」から成る。

アリスは、上述の図２ａに示されているような法的又は技術的保護のいずれかを用いて、ボブの攻撃から自らのコードを保護することができる。著作権法はソフトウエア工作物を確かに網羅している

ものの、アリスのような小さい会社がより大きくより力のある競合相手に法律を遵守させることは、経済的な現実から困難なことである。より魅力的な解決法は、アリスが、リバースエンジニアリングを技術的に極めてむずかしいものとし、リバースエンジニアリングを不可能とし又は少なくとも経済的に実現するのがほとんど不可能となるようにすることによって、自らのコードを保護することにある。技術的保護におけるいくつかの初期の試みがGoslerによって記載されている。（James R. Gosler, Software protection:Myth or reality? In CRYPTO'85…Advances in Cryptology, pages 140-157, 1985年8月）。

最も安全なアプローチは、アリスが自らのアプリケーションを全く販売せず、むしろそのサービスを売ることである。換言すると、ユーザはアプリケーション自体にアクセスできることは決してなく、むしろ毎回少額の電子通貨を支払って図３ａに示されているように、遠隔でプログラムを走らせるためにアリスのサイトに接続する。アリスにとっての利点は、ボブがそのアプリケーションに対する

物理的アクセスを獲得することは決してなく、従ってそれをリバースエンジニアリングできないということにある。そのマイナス面は当然のことながら、ネットワーク帯域幅及び待ち時間に関する限界のため、アプリケーションが、ユーザのサイト上で局所的に実行された場合よりもはるかに悪い性能を示す可能性があるということにある。部分的解決法は、アプリケーションを２つの部分、すなわち図３ｂに示されているようにユーザのサイト上で局所的に走らせる公開部分及び遠隔で走らせる（アリスが保護したいアルゴリズムを含む）私用部分に分割することである。

もう１つのアプローチは、自らのコードを例えば図４ａに示されているように、ユーザに送る前にアリスがそのコードを暗号化する

ことであろう。残念なことにこれは、解読／実行プロセス全体がハードウェア内で行なわれる場合にのみ機能する。このようなシステムは、Herzberg（Amir Herzberg and Shlomit S. Pinter.Public protection of software. ACM Transactions on Computer Systems, 5(4):371-393, 1987年11月）及びWilhelm（Uwe G. Wilhelm. Cryptographically protected objects. http://lsewww. epfl. ch/-wilhelm/CryPO. html.１９９７）に記述されている。コードが仮想計算機インタプリタによりソフトウェア内で実行される場合（Java bytecodesが最も頻繁にそうであるように）、ボブが解読済みコードを傍受しデコンパイルすることはつねに可能となる。

Java™プログラミング言語は、主としてそのアーキテクチャニュートラルバイトコードのため人気を博した。これは明らかに移動コードを容易にするものの、ネーティブコードに比べ１ケタ分性能を低下させる。予想できた通りに、このことは、Javaバイトコードを実行中にネーティブコードに翻訳するジャストインタイムコンパイラの開発を導いた。アリスは、全ての一般的なアーキテクチャについて自らのアプリケーションのネーティブコードバージョンを作成するべくこのような翻訳プログラムを使用することができた。アプリケーションをダウンロードするとき、ユーザのサイトは、それが走っているアーキテクチャ／オペレーティングシステムの組合せを識別しなくてはならず、例えば図４ｂに示されている

ように、対応するバージョンが伝送されることになるだろう。ネーティブコードにアクセスできるだけでは、ボブのタスクは、不可能ではないものの、さらにむずかしくなる。ネーティブコードを伝送する上での複雑性はさらに増す。問題は、実行前にバイトコード確認を受けるJavaバイトコードとは異なり、ネーティブコードはユーザの機械上で完全に安全に走行しえないということである。アリスが共同体の信

頼されたメンバーである場合、ユーザは、アプリケーションがユーザの側で何も有害なことはしないという彼女の保証を受諾することができる。誰れもアプリケーションを汚染しようとしていないことを確かめるためには、アリスは、コードが自らの書込んだオリジナルのコードであることをユーザに立証するべく、伝送中のコードにデジタル署名しなければならないだろう。

我々が考慮する最後のアプローチは、例えば図５に示されているようなコード混乱化である。基本的な考え方は、アリスが、アプリケーションをオリジナルと機能的には同一であるもののボブにとってはるかに理解し難いものであるアプリケーションに変換するプログラムである混乱化器を通して、自らのアプリケーションを走行させるというものである。我々は、混乱化が、それに値する注目を今後受けるはずであるソフトウエア取引秘密の保護のための実現可能な技術であると信じている。

サーバ側実行とは異なり、コード混乱化は、悪意あるリバースエンジニアリング努力から１つのアプリケーションを完全に保護することは決してできない。充分な時間と決意を与えられたボブは、その重要なアルゴリズム及びデータ構造を検索するためアリスのアプリケーションを吟味することが常に可能である。この努力を助けるべく、ボブは、混乱化変換を取り消すことを試みる自動混乱化解除プログラムを通して、混乱化済みコードを走行させようとするかもしれない。

従って、混乱化器がアプリケーションに付加するリバースエンジニアリングからのセキュリティレベルは、例えば、（ａ）混乱化器によって利用される変換の精巧化、（b）利用可能な混乱化解除アルゴリズムのパワー及び(c)混乱解除器が利用できる資源（時間及び空間）の量によって左右される。理想的には、暗号化

（大きい素数

を発見することが容易）及び解読（大きい数をファクター化するのが困難）のコストの劇的な差が存在する現在の公開かぎ暗号方式における状況を模倣したいと考える。実際そこには、以下で議論するようにポリノミナルタイム（多項式的時間）において応用できるものの、混乱化解除するのにイクスポネンシャルタイム（指数的時間）を必要とする混乱化変換が存在することになるだろう。

３．Java混乱化器の設計

図６は、Java混乱化器であるKavaのアーキテクチャを示す。ツールに対する主要入力は、Javaクラスファイルセット及びユーザが要求する混乱化レベルである,。ユーザは任意には、Javaプロファイリングツールによって生成されるとおり、プロファイリングデータのファイルを提供することができる。この情報は、アプリケーションのうち頻繁に実行される部分が非常に高価な変換によって混乱化されないことを確認するべく混乱化器を案内するのに使用可能である。ツールに対する入力は、Javaクラスファイルの１セットとして与えられるJavaアプリケーションである。ユーザは同様に、必要とされる混乱化レベル（例えば効力）及び混乱化器がアプリケーションに付加することを許されている最大実行時間/空間ペナルティ（コスト）を選択する。Kavaは、直接又は間接的に参照指示されたあらゆるライブラリファイルと共にクラスファイルを読みとりパージングする。完全に継承ツリーが、全てのシンボルについてのタイプ情報を示すシンボルテーブル及び全てのメソッドについての制御フローグラフと共に構築される。

Kavaは、以下で記述する大きなコード変換プールを含んでいる。しかしながら、これらが適用可能となる前に、前処理用バスが、１つの実施形態に従って、アプリケーションについてのさまざまなタ

イプの情報を収集する。一部の種類の情報は、手順間データフロー分析及びデータ従属性分析といった標準コンパイラ技術を用いて集めることができ、又ユーザによって提供されうるものも、又専門化された技術を用いて集められるものもある。例えば、プラグマティック分析は、どんな種類の言語コンストラクト及びプ

ログラミングイディオムがそれを含んでいるかをみるためアプリケーションを分析する。

前処理用パスの間に集められた情報は、適切なコード変換を選択し適用するために用いられる。アプリケーション内のあらゆるタイプの言語コンストラクトが、混乱化の対象でありうる。例えば、クラスを分割又は併合することができ、メソッドを変更又は作成することができ、又新しい制御及びデータ構造を作成すること及びオリジナルのものを修正することが可能である。アプリケーションに付加される新しいコンストラクトは、前処理パス中に集められたプラグマティック情報に基づいて、ソースアプリケーション内のものにできるかぎり類似したものとなるように選択することができる。

変換プロセスは、必要とされる効力が達成されたか又は最大コストを上回ってしまうまで反復される。ツールの出力は、機能的にオリジナルのものと等価である、Javaクラスファイルセットとして通常与えられる新しいアプリケーションである。ツールは同様に、変換がそれについて適用された情報、及び混乱化済みコードがオリジナルコードといかに関係しているかの情報が注釈として付けられたJavaソースファイルを生成することもできるだろう。注釈のついたソースは、デバッキングのために有用となる。

４．混乱化変換の分類

この好ましい実施形態の詳細な説明の残りの部分で、さまざまな

混乱化変換について記述し、それを分類し評価する。まず混乱化変換の概念を形成化することから始める。

定義１（混乱化変換）

$P -T-> P'$を法的混乱化変換とし、ここにおいて、以下の条件が保たれなくてはならない。

－　Pが終結できないか又はエラー条件で終結した場合、P’は終結してもしなくてもよい。

－　そうでない場合、P’は終結し、同じ出力をPとして生成しなければならない。

可観測性挙動は、あいまいに「ユーザが経,験する通りの挙動」として定義される。これはすなわち、ユーザがその副作用を経験しないかぎり、Ｐ’は、Ｐがもたない副作用（例えばファイルの作成又はインタネット上でのメッセージ送付）を持ちうる、ということを意味する。我々はＰ及びＰ’が同じように効率のよいものであることを要求していないという点に留意されたい。実際、我々の変換のうちの多くのものの結果として、Ｐ’はＰよりも緩慢になったり、Ｐよりも多くのメモリを使用することになる。

混乱化技術の異なるクラス間での主な分割ラインが図２ｃに示されている。まず第１に、それが目標とする情報の種類に従って混乱化変換を分類する。いくつかの単純な変換は、ソースコードフォーマティングといったアプリケーションの語い構造（レイアウト）及び変数の名前を目標とする。１つの実施形態においては、興味の対象であるより精巧な変換は、アプリケーションによって用いられるデータ構造又はその制御フローのいずれかを目標とする。

第２に、それが目標とされた情報について実行するオペレーションの種類に応じて変換を分類する。図２ｄ～２ｇを見ればわかるように、対照又はデータの集合を操作する複数の変換が存在する。か

かる変換は標準的にプログラマによって作成された抽象化を分解するか又は、関係のないデータ又は制御を合わせて束にすることによって新しい偽りの抽象を構築する。

同様にして、いくつかの変換は、データ又は制御のオーダリングに影響を及ぼす。数多くの場合において、２つの項目が宣言されるか又は２つの計算が行なわれる順番は、プログラムの可観測性挙動に対しいかなる効果ももたない。しかしながら、プログラムを書込んだプログラマならびにリバースエンジニアにとって、選択された順序で埋め込まれたはるかに有用な情報が存在しうる。空間的又は時間的に２つの項目又は事象が近ければ近いほど、それらがいずれかの形で関係をもつ確率は高くなる。オーダリング変換は、宣言又は計算の順序をランダム化することによりこれを調査しようとする。

５．混乱化変換の評価

いずれかの混乱化変換の設計を試みる前に、かかる変換の質を評価することができなくてはならない。この節では、複数の基準すなわち、それらがどれほどのあいまいさをプログラムに付加するか（例えば効力）、混乱化解除器にとってそれらがいかに破壊し難いものであるか（例えば弾力性）そして、混乱化済みアプリケーションに対しそれらがどれほどの計算オーバヘッド（例えば、コスト）を付加するかという基準に従って変換を分類することを試みる。

５．１　効力の尺度

まず最初に、プログラムＰ’にとって、プログラムＰよりもさらにあいまい（又は複雑又は読取り不能）であるということが何を意味するかを定義づけする。このような全てのメトリックは、定義上、それが人間の認識能力に（一部）基づかなくてはならないことか

ら比較的不確かなものでありうる。

幸いなことに、ソフトウエアエンジニアリングのソフトウエア複雑性メトリック分岐における多くの研究を利用することができる。この分野においては、メトリックは、読取り可能で、信頼性が高く維持できるソフトウエアの構築を助けることを意図して設計される。メトリックは往々にしてソースコードのさまざまなテクチュア特性を計数しこれらの計数値を複雑性の尺度へと組合わせることに基づいている。これまでに提案されてきた公式のいくつかは、実際のプログラムの実験的研究から誘導されてきたが、その他のものは純粋に投機的なものであった。

メトリックの文献中に見られる詳細な複雑性の公式は、「プログラムＰ及びＰ’が、Ｐ’がＰに比べより多くの特性ｑを含んでいるという点を除いて同一である場合、Ｐ’はＰよりもさらに複雑である」といったような一般的文を誘導するのに使用することができる。このような文が与えられた場合、我々は、これがそのあいまいさを増大する可能性が高いことを知りながら、プログラムに対しより多くのｑ－特性を付加する変換を構築することを試みることができる。

図７は、Ｅ（Ｘ）がソフトウエア構成要素Ｘの複雑性であり、Ｆが関数又は方法であり、Ｃがクラスであり、Ｐがプログラムである、より評判の良い複雑性尺

度のいくつかを作表したテーブルである。ソフトウエア構築プロジェクトにおいて使用された場合、標準的な最終目的はこれらの尺度を最小にすることである。これとは逆に、プログラムを混乱化する場合、我々は一般的に、尺度を最大にしたいと考える。

複雑性メトリックスにより我々は効力という概念を形式化することができ、これは以下で変換の有用性の尺度として使用されること

になる。非公式には、１つの変換は、それがアリスのオリジナルコードの意図を隠すことによってボブを混乱させる優れた働きをする場合に効力がある。換言すると、変換の効力は、オリジナルコードに比べ（人間にとって）混乱化済みコードがどれほど理解し難いかを測定する。これは、以下の定義において形式化される：

定義２（変換効力）Ｔを挙動保存変換とし、$P -^{T}-> P'$がソースプログラムを目標プログラムＰ’に変換するようにする。Ｅ（Ｐ）を、図７のメトリックの１つにより定義される通りのＰの複雑性とする。

$T_{pot}$（Ｐ）すなわちプログラムＰに対するＴの効力は、ＴがＰの複雑性を変更する程度の尺度である。これは、

$$T_{pot}(P) \stackrel{def}{=} E(P') / E(P) - 1$$

として定義される。Ｔは、$T_{pot}$（Ｐ）＞０である場合、効力ある混乱化変換である。

この議論においては、３点目盛（低、中、高）上で効力が測定される。

テーブル１中の観察事項は、我々が変換Ｔのいくつかの望ましい特性をリストアップすることを可能にする。Ｔが効力ある混乱化変換であるためには、それが以下のことを行なうことが必要である。

－ プログラムサイズ（$u_1$）全体を増大させ、新しいクラス及び方法（$u^a_7$）を導入する。

－ 新しい述語（$u_2$）を導入し条件付き及びルーピングコンストラクトのネスティングレベル（$u_3$）を増大させる。

－ メソッド引き数の数（$u_5$）及びクラス間インスタンス変数従属性

（$u^{d}_{7}$）を増大させる。

－　継承ツリーの高さ（$u^{b,c}_{7}$）を増大させる。

－　長範囲変数従属性（$u_{4}$）を増大させる。

## ５．２　弾力性の尺度

一見して、$T_{pot}$（P）を増大させることがトリビアルであると思われる。例えば$u_{2}$メトリックを増大させるために、我々がなすべきことはPに対しいくつかの任意のif文を付加することだけである：

```
main() {                    main()  {
S1;                         S1;
S2;     = T =>              if (5==2) S1;
                                  S2;}
                            if (1>2) S2;
                          }
```

残念なことに、このような変換は、単純な自動技術によって容易に取り消されうることから、事実上役に立たない。従って、自動混乱解除器の攻撃下でどれほど変換が耐えられるかを測定する弾力性の概念を導入することが必要である。例えば、変換Tの弾力性は、次の２つの尺度の組合せとして見ることができる：すなわち

プログラマの努力：Tの効力を有効に低減させることができる自動混乱化解除器を構築するのに必要とされる時間の量：及び

混乱解除器の努力：かかる自動混乱解除がTの効力を有効に低減させるのに必要とされる時間及び空間。

弾力性と効力を区別することが重要である。変換は、それが人間の読み手を混乱させようとする場合には効力があるが、自動混乱化解除器を混乱させる場合には弾力性がある。

弾力性は、図８ａに示されているように、トリビアルからワンウエイまでの目盛上で測定される。ワンウェイ変換は、それらが決して取り消され得ないという意味で特殊である。これは標準的に、これらの変換が人間のプログラマにとって

は有用であったがプログラ

ムを正しく実行するためには必要でないプログラムからの情報を除去するからである。例としては、フォーマティングを除去し、変数名をスクランブルする変換が含まれる。

その他の変換は、標準的には、その可観測性挙動を変更しないが、人間の読み手に対する「情報負荷」を増大させるような役に立たない情報をプログラムに付加する。これらの変換は、変動する困難度で取り消しが可能である。

図８ｂは、混乱化解除器の努力がポリノミアルタイム又はイクスポネンシャルタイムのいずれかとして分類されることを示す。プログラマの努力、つまり変換Ｔの混乱化解除を自動化するのに必要とされる作業は、Ｔの範囲の関数として測定される。これは、プログラム全体に影響を及ぼしうるものに対してよりも手順のうちの小さな部分のみに影響を及ぼす混乱化変換に対する対策を構築する方が容易であるという直観力に基づいている。

変換の範囲は、コード最適化理論から借りた用語を用いて定義される：すなわち、Ｔは、それが制御フローグラフ（ＣＦＧ）の単一基本ブロックに影響を及ぼす場合、局所変換であり、ＣＦＧ全体に影響を及ぼす場合大域変換であり、手順間の情報フローに影響を及ぼす場合手順間変換であり、それが独立して実行する制御スレッド間の相互作用に影響を及ぼす場合プロセス間変換である。

定義３（変換弾力性）Ｔを挙動保存変換とし、Ｐ＝Ｔ＝＞Ｐ’がソースプログラムＰを目標プログラムＰ’に変換するようにする。$T_{res}$（Ｐ）は、プログラムＰに対するＴの弾力性である。$T_{res}$（Ｐ）は、ＰがＰ’から再構築され得ないようにＰから情報が除去される場合、ワンウェイである。そうでなければ、

$T_{res} \stackrel{def}{=}$弾力性（$T_{DEobfuscator\ effort}$，$T_{programmer\ effort}$）であり、ここで弾力性は、図８ｂ中でマトリクス内に定義

された関数である。

５．３　実行コストの尺度

図２ｂでは、効力と弾力性が、１つの変換を記述する３つの構成要素のうちの

２つであることがわかる。第３の構成要素すなわち変換のコストは、変換が混乱化済みアプリケーションに対して招来する実行時間又は空間ペナルティである。我々は、コストを４点目盛（無料、安価、高価、法外）上で分類し、このうちの各評点について以下で定義する。

定義５（変換コスト）Ｔを挙動保存変化とし、$T_{cost}$（Ｐ）∈｛法外、高価、安価、無料｝となるものとし、Ｐ’の実行がＰよりも０（１）個多い資源を必要とする場合$T_{cost}$（Ｐ）＝無料；Ｐ’の実行がＰよりも０（ｎ）個多い資源を必要とする場合、$T_{cost}$（Ｐ）＝安価；Ｐ＞１でＰ’の実行がＰよりも０（$n^P$）個多い資源を必要とする場合、$T_{cost}$（Ｐ）＝高価；又そうでなければ、$T_{cost}$（Ｐ）＝法外（すなわちＰ’の実行がＰよりも指数的に大きい資源を必要とする）とする。

交換に付随する実際のコストが、その適用環境により左右されることに留意すべきである。例えば、プログラムの最上レベルに挿入された単純な割当て文ａ＝５は、恒常なオーバヘッドしかこうむらない。内部ループ内部に挿入された同じ文は、実質的にさらに高いコストを有することになる。他の指摘のないかぎり、我々につねに、それがソースプログラムの最も外側のネスティングレベルで適用されたかのように変換のコストを提供する。

５．４　質の尺度

ここで、混乱化変換の質の正式の定義を示すことができる：

定義６（変換の質）

変換Ｔの質$T_{qual}$（Ｐ）は、Ｔの効力、弾力性及びコストの組合せとして定義される：$T_{qual}$（Ｐ）＝（$T_{pot}$（Ｐ），$T_{res}$（Ｐ），$T_{cost}$（Ｐ））。

５．５　レイアウト変換

新しい変換について調査する前に、例えばCremaといった現行のJava混乱化器に典型的であるトリビアルレイアウト変換について簡単に見ていく。(Hans Peter Van Vliet. Crema……Java混乱化器。http.//web.inter.nl.net/users/H.P.van.Vliet/crema,html，１９９６年１月）。第１の変換は、Javaクラスファイル内

で時として利用可能なソースコードフォーマティング情報を除去する。これは、オリジナルフォーマッティングがひとたび過ぎ去るとそれを回復できないことから、ワンウェイ変換である；フォーマッティングにおいてきわめてわずかな意味論的内容しか存在せず、その情報が除去された時点で大きな混乱は全く導かれないことから、これは低効力の変換である,最後に、これは、アプリケーションの空間及び時間的複雑性に影響が及ぼされないことから、無料の変換である。

識別子名のスクランブリングも同様にワンウェイでかつ無料の変換である。しかしながら、それは、識別子がプラグマティック情報を多く含んでいることから、フォーマッティング除去よりもはるかに高い効力をもつ。

６．制御変換

本節及び以下の数節では、混乱化変換のカタログを紹介する。そのいくつかは、コンパイラ最適化及びソフトウエアリエンジニアリングといった他の分野で使用された周知の変換から誘導されたものであり、他のものは本発明の１実施形態に従った混乱化のみを目的として開発されたものである。

本節では、ソースアプリケーションの制御フローをあいまいにしようとする変換について論述する。図２ｆに示されているように、我々はこれらの変換を、制御の流れの集合、オーダリング又は計算に影響を与えるものとして分類する。制御集合変換は、論理的に相互帰属計算を分割するか又は共に属さない計算を併合する。制御オーダリング変換は、計算が実施される順序をランダム化する。計算変換は新しい（冗長又はデッド）コードを挿入するか又はソースアプリケーションに対しアルゴリズム変更を行なうことができる。

制御フローを変える変換については、一定量の計算オーバヘッドが不可避であろう。アリスにとってはこれは、彼女が非常に効率の良いプログラムときわめて混乱化されたプログラムの間での選択をせまられる可能性があることを意味している。混乱化器がこのトレードオフにおいて、安価な変換と高価な変換の間での選択が行なえるようにすることによって彼女を補助することができる。

６．１　不明瞭な述語

制御変更変換を設計するときの実際の課題は、それらを安価にすることだけで

なく、混乱化解除器からの攻撃に対し耐性あるものにすることにもある。これを達成するため、数多くの変換は、不明瞭変数及び不明瞭述語の存在に依存している。非公式には、変数Ｖは、それが先験的に混乱化器に知られているものの混乱化解除器が演繹しがたいいくつかの特性ｑを有する場合、不明瞭である。同様にして、混乱化器には周知であるもののその成果を演繹することが混乱化解除器にとってきわめて困難である場合、その述語Ｐ（論理式）は不明瞭である。

混乱化解除器にとって解明がむずかしい不明瞭な変数及び述語を作成できることが、混乱化ツールの作成者にとって主要な挑戦であり、きわめて弾力性のある制御変換への鍵である。我々は、不明瞭

な変数又は述語の弾力性（即ち混乱化解除攻撃に対するその耐性）を変換弾力性と同じ目盛上で測定する（すなわちトリビアル、弱、強、フル、ワンウェイ）。同様にして我々は、不明瞭なコンストラクトの付加コストを変換コストと同じ目盛（すなわち、無料、安価、高価、法外）上で測定する。

定義７（不明瞭コンストラクト）１つの変数Ｖは、それが、混乱化時点で知られている点ｐにおける特性ｑを有する場合、プログラム中の点ｐにおいて不明瞭である。ｐが文脈から明白である場合、我々はこれを$V_q^p$又は$V_q$と書く。述語Ｐは、その成果が混乱化時点で知られている場合、ｐにおいて不明瞭である。我々は、Ｐがｐにおいてつねに、偽（真）に評価する場合$P^F_p$（$P^T_p$）と書き、Ｐが時に真、時に偽に評価する場合、$P^?_p$と書く。ここでも又、ｐは、文脈から明白である場合省略されることになる。図９は、異なるタイプの不明瞭述語を示す。実線は、時としてとられ得る経路を表わし、破線は、決してとられることのない経路を示す。

以下では、単純な不明瞭コンストラクトのいくつかの例を示す。これらは、混乱化器にとって構築しやすく、混乱化解除器にとって同様に解読しやすいものである。第８節は、はるかに高い弾力性をもつ不明瞭コンストラクトの例を提供する。

６．１．１　トリビアル及び弱不明瞭コンストラクト

不明瞭コンストラクトは、混乱化解除器が統計的局所分析によりそれを解読で

きる（すなわちその値を演繹できる）場合、トリビアルである。分析は、それが制御フローグラフの単一基本ブロックに制限される場合、局所的である。図１０ａ及び１０ｂは、(a) トリビアル不明瞭コンストラクト及び(b)弱不明瞭コンストラクトの例を提供している。

我々は同様に、１つの不明瞭変数が呼出しから単純な良く理解さ

れている意味論を用いてライブラリ関数へと計算される場合、この変数をトリビアルであるとみなす。標準的なライブラリクラスセットを支持するのに全ての実施を必要とする言語であるJava™のような言語については、かかる不明瞭変数は容易に構築される。単純な例は、ランダム（ａ，ｂ）が、ａ・・・ｂの範囲内の１つの整数を戻すライブラリ関数であるint $V^S$〔１，５〕－ランダム（１，５）である。残念なことに、かかる不明瞭変数は同様に混乱化解除が容易である。必要なのは、全ての単純なライブラリ関数の意味論を混乱化解除器設計者が作表し次に混乱化されたコード内で関数呼出しについてパターン照合することだけである。

不明瞭コンストラクトは、静的大域分析により混乱化解除器がそれを解読できる場合、弱である。分析は、それが単一の制御フローグラフに制限されている場合に大域的である。

６．２　計算変換

計算変換は、次の３つのカテゴリに入る：すなわち、実際の計算に寄与しない無関係の文の後ろに実の制御フローを隠す；対応する高レベルの言語コンストラクトが全く存在しないオブジェクトコードレベルでコードシーケンスを導入する、又は実の制御フロー抽象を除去するか又はスプリアスなものを導入する。

６．２．１　デッドコード又は無関係コードの挿入

$U_2$及び$U_3$メトリックは、一片のコードの感知された複雑性とそれが含む述語の数の間に強い相関関係が存在することを示唆している。不明瞭述語を用いると、プログラム内に新しい述語を導入する変換を考案することができる。

図１１中の基本ブロックＳ＝$S_1$……$S_n$を考慮する。図１１ａ中では、不明瞭言語$P^T$をＳ内に挿入して、基本的にそれを半分に分割する。$P^T$述語は、それが

つねに真に評価することになるため

、無関係コードである。図１１ｂでは、ここでも又Ｓを２つの半分に分割し、次に、この第２の半分の２つの異なる混乱化済みバージョン$S^a$及び$S^b$を作成するべく進む。$S^a$及び$S^b$は、Ｓの第２の半分に対して異なるＯＢＳ変換セットを適用することによって作成されることになる。従って、リバースエンジニアにとって、$S^a$及び$S^b$が実際に同じ機能を果たすことは直接明白ではない。我々は、ランタイムで$S^a$及び$S^b$の間での選択を行なうために述語$P^7$を使用する。

図１１ｃは、図１１ｂと類似しているが、今度は、$S^b$内にバグを導入する。$P^T$述語はつねに、コードの正しいバージョン$S^a$を選択する。

６．２．２　ループ条件の拡張

図１２は、終結条件をより複雑なものにすることによっていかにしてループを混乱化できるかを示している。基本的な考え方は、ループが実行する予定の回数に影響を及ぼさない$P^T$又は$P^F$述語でループ条件を拡張するということである。例えば図１２ｄで我々が付加した述語は、$X^2(X+1)^2=0 \pmod 4$であることからつねに真に評価される。

６．２．３　可約から非可約フローグラフへの変換

往々にして、プログラミング言語は、言語自体よりもさらに表現力のあるネーティブ又は仮想計算機コードにコンパイルされる。これがあてはまる場合、こうして我々は言語分割変換を考案できるようになる。変換が言語分割変換であるのは、それがいずれのソース言語コンストラクトとも直接的対応性を全くもたない仮想計算機（又はネーティブコード）命令シーケンスを導入する場合である。かかる命令シーケンスと直面した場合、混乱化解除器は、等価（ただし重畳された）ソース言語プログラムを合成することを試みるか又

は全て放棄しなければならなくなる。

例えば、Java™バイトコードは、ＧＯＴＯ命令を有するがJava™言語は、対応するＧＯＴＯ文を全くもたない。このことはすなわち、Java™バイトコードが任意の制御フローを表現でき、一方Java™言語は単に構造化された制御フローしか

（容易に）表現できないということを意味している。標準的には、Java™プログラムから生成された制御フローグラフがつねに可約となるものの、Java™バイトコードは、非可約フローグラフを表現することができると言うことができる。

非可約フローグラフの表現は、ＧＯＴＯが無い言語ではきわめて扱いにくくなるため、我々は可約フローグラフを非可約フローグラフに転換する変換を構築する。これは、構造化されたループを、多重ヘッダーのあるループへと変えることによって行なうことができる。例えば、図１３ａでは、不明瞭述語$P^F$をWhileループに加えて、そのループの中央への飛越しが存在することがわかるようにする。事実、この分岐は決してとられることがない。

Java™デコンパイラが、コードを複製するものか又は外来のブール変数を含むものへと非可約フローグラフを変えなければならなくなる。代替的には、混乱化解除器が、混乱化器によって全ての非可約フローグラフが生成されたことを推測し、単に不明瞭述語を除去することが可能となる。これに対抗するため、時として図１３ｂに示された代替的変換を使用することができる。混乱化解除器が$P^F$を盲目的に除去した場合、結果として得られるコードは正しくなくなる。

特に、図１３ａ及び１３ｂは、可約フローグラフを非可約フローグラフに変換するための変換を例示している。図１３ａでは、ループ本体Ｓ２を２つの部分（$S^a{}_2\nabla$及び$S^b{}_2$）に分割し、偽りの

飛越しを、$S^b{}_2$の始めに挿入する。図１３ｂでは、同じくＳ１を２つの部分$S^a{}_1$及び$S^b{}_1$に分割する。$S^b{}_1$はループ内に移動させられ、不明瞭述語$P^T$が、$S^b$がつねにループ本体の前に実行されることを確実にしている。第２の述語$Q^F$は、$S^b{}_1$が１度だけ実行されることを確実にしている。

６．２．４　ライブラリ呼出し及びプログラミングイディオムの除去

Javaで書き込まれる大部分のプログラムは、標準ライブラリに対する呼出しに大きく依存している。ライブラリ関数の意味論は周知であることから、かかる呼出しは、リバースエンジニアに対して有利な手掛りを提供する可能性がある。Javaライブラリクラスに対する参照指示がつねに名前によるものでありこれらの名前を混乱化することはできないという事実によって問題は悪化する。

数多くの場合、混乱化器は、単純に標準ライブラリのその独自のバージョンを提供することによって、これに対抗することができるだろう。例えば、（ハッシュテーブル実施を使用する）Javaディクショナリクラスへの呼出しは、同一の挙動を伴うものの例えば赤－黒のツリーとして実施されたクラスに対する呼出しへと変えられ得る。この変換のコストは、実行時間についてはさほど大きくないがプログラムのサイズについては大きい。

類似の問題が、数多くのアプリケーションで頻繁に発生する共通のプログラミングイディオムであるクリシエ（又はパターン）でも発生する。経験豊かなリバースエンジニアは、見慣れないプログラムについての自からの理解を飛越し－開始するためにかかるパターンをサーチするだろう。一例として、Java™内のリンクされたリストを考慮する。Java™ライブラリは、挿入、削除及び列挙といった共通リストオペレーションを提供する標準クラスを全くもたない。

その代り、大部分のJava™プログラマは、それらを次のフィールド上で合わせてリンクすることにより、特別のものとしてオブジェクトリストを構築することになる。かかるリストを通しての反復は、Java™プログラムにおいては非常に共通したパターンである。自動化されたプログラム認識の分野で発明された技術（本書に参考とし内含されているLinda Mary Wills,自動化プログラム認識；フィージビリティの立証、人工頭脳、４５（１－２）；１１３－１７２，１９９０参照を参照のこと）が、共通パターンの識別及びさほど明白でないものとの置換のために使用可能である。例えばリンクされたリストの場合には、要素アレイ内にカーソルといったさほど共通でないもので標準リストデータ構造を表わすかもしれない。

６．２．５　テーブル解釈

最も有効な（かつ高価な）変換の１つは、テーブル解釈である。考え方は、コードの１区分（この例ではJavaバイトコード）を異なる仮想計算機コードに転換するというものである。この新しいコードはこのとき、混乱化済みアプリケーションと共に内含された仮想計算機インタプリタにより実行される。明らかに、特定のアプリケーションには、各々異なる言語を受諾し混乱化済みアプリケーショ

ンの異なる区分を実行する複数のインタプリタが含まれている可能性がある。

各々の解釈レベルについて通常１桁の減速が存在することから、この変換は、全体的ランタイムの小さな部分を構成するか又は非常に高レベルの保護を必要とするコードの区分に予約されたものであるべきである。

６．２．６　冗長オペランドの付加

ひとたびいくつかの不明瞭変数を構築したならば、算術式に冗長オペランドを付加するために代数法則を使用することができる。こ

うして、$U_1$メトリックが増大されることになる。明らかにこの技術は、数値的な制度が問題でない整数式の場合に最もうまく機能する。以下の混乱化済みの文（１’）では、値が１である不明瞭変数Ｐが使用される。文（２’）では、値が２である不明瞭部分式Ｐ/Ｑを構築する。明らかに、文（２’）に達した時につねにそれらの商が２となるかぎりにおいて、プログラム実行中に、Ｐ及びＱに異なる値をとらせることができる。

（１）　Ｘ＝Ｘ＋Ｖ；$=^{T}=>$　（１’）　Ｘ－Ｘ＋Ｖ＊$P^{-1}$；

（２）　Ｚ＝Ｌ＋１；　　（２’）　Ｚ＝Ｌ＋（$P^{=2Q}$／$Q^{=P/2}$）／２

６．２．７　コードの並列化

自動並列化は、マルチプロセッサ計算機上でランするアプリケーションの性能を増大させるのに用いられる重要なコンパイラ最適化である。我々がプログラムを並列化することを望む理由は、当然のことながらさまざまである。我々は、性能を増大させるためではなく、実際の制御フローをあいまいにするために並行性を増大することを望んでいる。考えられる２つのオペレーションが利用可能である：すなわち、

１．有用なタスクを全く行なわないダミープロセスを作成することができる。又

２．アプリケーションコードの逐次区分を、並行して実行する多重区分へと分割することができる。

アプリケーションが単一プロセッサ計算機上でランしている場合、我々はこれらの変換が多大な実行時間ペナルティを有することを予想することができる。こ

れは、これらの変換の弾力性が高いことから、数多くの状況において受諾できるものである。すなわち、プログラムを通しての考えられる実行経路の数が実行プロセス数と共

に指数的に大きくなることから、並列プログラムの静的分析は非常にむずかしい。並列化は又、高レベルの効力をも生み出す：すなわち、リバースエンジニアは、逐次プログラムに比べ、並列プログラムがはるかに理解し難いものであることがわかるだろう。

図１４に示されているように、１コード区分は、それがデータ従属性を全く含まない場合、容易に並列化されうる。例えば、$S_1$及び$S_2$が２つのデータ独立型文である場合、これらは並行してランされ得る。Java™言語といったような明示的な並列コンストラクトを全くもたないプログラミング言語においては、プログラムはスレッド（軽量プロセス）ライブラリに対する呼出しを用いて並列化され得る。

図１５で示されているように、データ従属性を含むコード区分は、アウェイアンドアドバンスといった適切な同期化基本命令を挿入することにより、同時並行スレッドへと分割され得る（本書に参考として内含されているMichael Wolfe, 並列計算用高性能コンパイラ. Addison-Wesley, １９９６。ＩＳＢＮＯ－８０５３－２７３０－４を参照のこと）。このようなプログラムは基本的に逐次的にランしているが、制御フローは、１つのスレッドから次のスレッドへとシフトしていることになる。

６．３　集合変換

プログラマは、抽象化を導入することによってプログラミングの固有の複雑性を克服する。１つのプログラムの数多くのレベルで抽象化が存在するが、手順抽象化は最も重要なものである。このような理由から、混乱化器にとっては、手順及びメソッド呼出しをあいまいにすることが重要であり得る。以下では、メソッド及びメソッド呼出しをあいまいにすることのできるいくつかの方法、すなわちインライン処理、アウトライン処理、インタリーブ及びクローニン

グについて考慮する。これら全ての背後にある基本的考えは、同じである：すなわち（1）プログラマが１つの方法に集合した（恐らくはそれが論理的に相互帰属していたため）コードは分割され、プログラム全体にわたり分散されるべきであること、及び(2)相互に帰属していないと思われるコードは１つのメソッドの形に集合させられるべきであることである。

６．３．１　インライン及びアウトライン方法

インライン処理は、当然のことながら、重要なコンパイラ最適化である。これは、プログラムから手順抽象を除去することから、きわめて有用な混乱化変換でもある。インライン処理は、手順呼出しがひとたび、呼出された手順の本体と置換され、手順自体が除去された場合、コード内には抽象の痕跡が全く残らないことから、きわめて弾力性の高い変換である（これは基本的にワンウェイである）。図１６は、手順Ｐ及びＱがその呼出しサイトでいかにインライン処理され、次にコードから除去されるかを示す。

アウトライン処理（一続きの文をサブルーチンへと変えさせること）は、インライン処理に対する非常に有用な姉妹変換である。我々は、Ｑのコードの始まり及びＰのコードの終りを新しい手順Ｒへと抽出することにより偽りの手順抽象を作成する。

Java™言語といったようなオブジェクト指向言語では、インライン処理は、事実上、つねに完全にワンウェイの変換であるとはかぎらない。メソッド呼出しｍ．Ｐ()を考えてみよう。呼出される実際の手順は、ｍのランタイムタイプによって左右されることになる。複数のメソッドを特定の呼出しサイトで呼出すことができる場合、我々は考えられる全てのメソッドをインライン処理し（本書に参考として内含されているJeffrey Dean, オブジェクト指向言語の全プログラム最適化。ワシントン大学博士論文、１９９６を参照の

こと）、ｍ）ａタイプについて分岐することにより適切なコードを選択する。従って、メソッドのインライン処理及び除去の後でさえ、混乱化済みコードはなおも、オリジナル抽象の痕跡を幾分か含んでいる可能性がある。例えば、図１７は、メソッド呼出しのインライン処理を例示している。ｍのタイプを統計的に決定

できるのでないかぎり、m．P（ ）を結びつけることのできる考えられる全てのメソッドが呼出しサイトでインライン処理されなくてはならない。

６．３．２　インタリーブメソッド

インタリーブメソッドの検出は、重要で困難なリバースエンジニアリングタスクである。

図１８は、同じクラスで宣言された２つのメソッドをどのようにインタリーブできるかを示す。考え方は、メソッドの本体及びパラメータリストを併合し余分のパラメータ（又は大域変数）を付加して個々のメソッドに対する呼出しを弁別するというものである。理想的には、メソッドは共通のコード及びパラメータの併合を可能にするべく性質が類似しているべきである。図１８の場合がこれであり、ここではＭ１及びＭ２の第１のパラメータは同じタイプをもつ。

６．３．３　クローンメソッド

サブルーチンの目的を理解しようとするとき、リバースエンジニアは当然のことながらその署名及び本体を検査することになる。しかしながら、ルーチンの挙動を理解するのに同じように重要なのは、それが呼出しされている異なる環境である。我々は、実際にはそうでないのに異なるルーチンが呼出されつつあるように見えるようにするべくメソッドの呼出しサイトを混乱化することによって、このプロセスをさらに難しくすることができる。

図１９は、オリジナルコードに異なる混乱化変換セットを適用することによってメソッドの複数の異なるバージョンを作成することができる。我々は、ランタイムで異なるバージョン間で選択を行なうのにメソッドディスパッチを使用する。

メソッドクローニングは、図１１の述語挿入変換と類似しているが、ここでは、コードの異なるバーション間で選択を行なうのに不明瞭述語ではなくメソッドディスパッチを使用しようとしているという点が異なっている。

６．３．４　ループ変換

（特に）数値アプリケーションの性能を改善することを意図して数多くのループ変換が設計されてきた。広範な調査についてはBacon〔２〕を参照のこと。こ

れらの変換のいくつかは、図７に関して、上述した複雑性メトリックも増大させることから、我々にとって有用である。図２０ａに示されているようなループブロック化は、内部ループが、キャッシュ内にフィットするように相互作用空間を分割することによりループのキャッシュ挙動を改善するために使用される。ループアンロールは、図２０ｂに示されている通り、１回又は複数回ループの本体を複製する。コンパイル時点でループ境界がわかっている場合、そのループを全体的にアンロールすることができる。図２０ｃに示されているようなループ分裂は、複合本体を伴うループを同じ反復空間を伴う複数のループへと変える。

３つの変換は全て、ソースアプリケーションの合計コードサイズ及び条件数を増大させることから、$U_1$及び$U_2$メトリックを増大させる。ループブロック化変換は同様に、余分のネスティングを導入し、従って$U_3$メトリックも増大させる。

分離して適用された場合、これらの変換の弾力性はきわめて低い。混乱化解除器がアンロールされたループを再度ロールするのに多大な静的分析は必要でない。しかしながら変換が組合わされた時点で、弾力性は劇的に上昇する。例えば、図２０ｂの単純なループが与えられた場合、我々はまず最初にアンロールを適用し、次に分裂、そして最後にブロック化を適用することができる。結果として得たループをそのオリジナル形態に戻すには、混乱化解除器にとってかなりの量の分析が必要となるだろう。

６．４　オーダリング変換

プログラマは、その局所性を最大限にするように自らのソースコードを組織する傾向をもつ。考え方は、論理的に関係する２つの項目がソーステキスト内で同様に物理的に近い場合に、プログラムがさらに読取り、理解しやすくなるというものである。この種の局所性は、ソースの全てのレベルで機能する：例えば、式内の項の間、基本ブロック内の文、方法内の基本ブロック、クラス内の方法そしてファイル内のクラスの間に局所性が存在する。全ての種類の空間的局所性がＲＥに対し有用な手掛りを提供することができる。従って、可能な場合にはつねに、我々はソースアプリケーション中の任意の項目の配置をランダム化する。いく

つかのタイプの項目（例えばクラス内のメソッド）について、これはトリビアルである。その他のケースでは（例えば基本ブロック内の文）、どの再オーダリングが技術的に有効であるかを見極めるため、データ従属性分析（本書に参考として内含されている（David F. Bacon, Susan L. Graham, 及びOliver J. Sharp. 高性能計算のためのコンパイラ変換A CM Computing Surveys, ２６（４）：３４５－４２０，１９９４年１２月、http://www.acm.org/pubs/toc/Abstracts/０３６０－０３００/１９７４０６．html. 及びMichael Wolfe. 並列計算のための高性能コンパイラ。Addison-Wesley, １９９６．ISBN ０－８０５３－２７３０－４，を参照のこと）が行なわれる。

これらの変換は低い効力を有する（プログラムに多大なあいまいさを付加しない）が、その弾力性は高く、数多くのケースでワンウェイである。例えば、基本ブロック内の文の配置がランダム化された時点で、結果として得たコードにはもとの順序の痕跡は全く残っていないことになる。

オーダリング変換、第６，３，１節の「インライン－アウトライン」変換に対する特に有用な姉妹変換である。その変換の効力は、(1) 手順Ｐ内で複数の手順呼出しをインライン処理すること、(2) Ｐ内の文の順序をランダム化すること及び、(3) Ｐの文の隣接区分をアウトライン処理することによって増強され得る。このようにして、以前複数の異なる手順の一部であった無関係な文が、偽りの手順抽象内に合わせて入れられる。

いくつかのケースでは、例えば後向きにランさせることによって、ループを再オーダリングすることも可能である。このようなループ逆転変換は、高性能コンパイラにおいて共通である。（David F. Bacon, Susan L. Graham, 及びOliver J. Sharp. 高性能計算のためのコンパイラ変換ACM Computing Surveys, ２６（４）：３４５－４２０，１９９４年１２月、http.//www.acm.org/pubs/toc/Abstracts/０３６０－０３００/１９７４０６．html.）。

７．データ変換

本節では、ソースアプリケーションで使用されるデータ構造を混乱化する変換について説明する。図２ｅに示しているように、こうした変換は、データの記憶

、符号化、集合、または、オーダリングを行う変換として分類される。

７．１　記憶および符号化変換

多くの場合には、プログラム中の特定のデータ項目を記憶するた

めの「自然な」方法がある。例えば、１つの配列の各要素を繰り返すためには、反復変数として適切なサイズのローカル整変数を割り当てることが選択されるだろう。他の変数タイプが使用可能であるが、こうした他の変数タイプは自然さの点で劣るだろうし、恐らくは効率的にも劣ることだろう。

さらに、変数のタイプに基づいている特定の変数が有することが可能であるビットパターンの「自然な」解釈が存在する場合も多い。例えば、一般的に、ビットバターン「０００００００００００１１００」を記憶する１６ビット整変数が整数値「１２」を表すと仮定される。当然のことながら、これらは単なる取決めであり、他の解釈が可能である。

混乱化記憶変換は、動的データと静的データとのための不自然な記憶域クラスを選択することを試みる。同様に、符号化変換は、共通データタイプに関して不自然な符号化を選択しようとする。記憶変換と符号化変換とが組み合わせて使用される場合が多いが、場合によっては、これらの変換の各々が単独で使用されることも可能である。

７．１．１　符号化の変更

符号化変換の単純な事例として、$c_1$と$c_2$とが定数である場合に、$i_0 = c_1 * i + c_2$で整変数ｉを置き換える。効率のために、$c_1$を２の累乗として選択することが可能である。次の例では、$c_1 = 8$、$c_2 = 3$とすると、

```
{                          =  T  =>        {
int i=1;                                   int i=11;
while (i < 1000)                           while (i < 8003)
     . . . A[i] . . .;                       . . A[(i-3)/8] . . .;
    i++;                                       i+=8;
              }                                          }
```

当然のことながら、オーバフロー（および、浮動小数点変数の場合には、正確

度）の問題に対処する必要がある。問題の変数の範囲（この範囲は、静的解析法を使用することによってまたはユーザに質問することによって決定されることが可能である）のせいでオーバフローが発生しないということ、または、より大きな変数タイプに変更可能であるということを調べることが可能である。

一方では弾力性と効力との間のトレードオフ、他方では弾力性とコストとの間のトレードオフとがあり得る。上記例の$i_0 = c_1 + i + c_2$のような単純な符号化関数は僅かな追加の実行時間しか付加しないが、一般的なコンパイラ解析法(Michael Wolfe. High performance Compilers For Parallel Computing. Addison-Wesley, 1996. ISBN 0-8053-2730-4、および、David F. Bacon, Susan L. Graham and Oliver J. Sharp. Compiler transformations for high-performance computing. ACM Computing Surveys, 26(4):345-420, December 1994. http://www.acm.org/pubs/toc/Abstracts/0360-0300/197406.html)を使用して混乱解除されることが可能である。

７．１．２　変数のプロモート

特殊化された記憶域クラスからより汎用のクラスに変数をプロモートする単純な記憶変換が幾つか存在する。こうした記憶変換の効力と弾力性は一般的に低いが、他の変換と組み合わせて使用される場合には、極めて有効であり得る。例えば、Javaでは、整変数が整数オブジェクトにプロモートされることが可能である。同じことが、対応する「パッケージ化された」クラスを全てが有する他のスカラ型の場合に当てはまる。Java™が不要部分の整理をサポートするので、オブジェクトがすでに参照されなくなっている時には、そのオブジェクトが自動的に除去されるだろう。ここに、その一例があ

る。

```
{                                   {
int I=1;                            int i = new int(1);
while (i < 9)   = T =>  while (i.value < 9)
    ...A[i]...;                         ...A[i.value]...;
    i++;                                i.value++;
}                                   }
```

変数の寿命を変更することも可能である。最も単純なこうした変換は、独立した手続き呼出しの間で後で共有されるグローバル変数へとローカル変数を変更する。例えば、手続きPと手続きQの両方がローカル整変数を参照し、かつ、PとQとが両方とも同時にアクティブであることが不可能である（プログラムがスレッドを含まなければ、これは、静的呼出しグラフ(static call graph)を調べることによって求められることが可能である）場合には、変数がグローバル変数にされて、これらの手続きの間で共有されることが可能である。

```
void P()  {               int C;
int i; ...I...            void P()  {
                          ...C...
                           }
void Q()      = T =>      while (i.value<9)
int k;...k...             ...C...
}                          }
```

PとQとによって参照されるグローバルデータ構造の数が増大させられるので、この変換は、$u_5$メトリックを増大させる。

7．1．3　変数の分割

ブール変数と制限された範囲内の他の変数とが、２つ以上の変数

に分割されることが可能である。ｋ個の変数$p_1$，．．．，$p_k$に分割された変数VをV－［$p_1$，．．．，$p_k$］と記述する。典型的には、この変換の効力がｋとともに増大するだろう。残念なことに、変換コストもｋとともに増大し、したがって、ｋを２または３に制限することが一般的である。

タイプＴの変数Ｖがタイプ Ｕの２つの変数ｐ、ｑに分割されることを可能にするためには、３つの情報断片、すなわち、（１）対応するＶの値にｐとｑの値をマップする関数Ｆ（ｐ；ｑ）、（２）対応するｐ、ｑの値にＶの値をマップする関数ｇ（Ｖ）、（３）ｐとｑとに対する演算の観点からキャストされた新たな演算（タイプＴの値に対する基本演算に対応する）が、与えられることが必要である。この節の残りの部分では、Ｖがブール変数タイプであり、ｐとｑとが小整変数であると仮定する。

図２１ａは、分割ブール変数に関して行われることが可能である表現の選択を示す。この表は、Ｖがｐとｑとに分割されている場合に、および、プログラムの何らかのポイントでｐ＝ｑ＝０またはｐ＝ｑ＝１である場合に、それが、Ｖが偽であることに相当するということを示している。同様に、ｐ＝０かつｑ＝１、または、ｐ＝１かつｑ＝０が、真に相当する。

この新たな表現で示される場合には、様々な組込みブール演算（例えば、ＡＮＤ、ＯＲ）に関する置換えが考案されなければならない。１つのアプローチは、各々の演算子に関する実行時ルックアップテーブルを提供することである。「ＡＮＤ」と「ＯＲ」とに関するテーブルが図２１ｃと図２１ｄとに別々に示されている。２つのブール変数Ｖ１＝［ｐ，ｑ］とＶ２＝［ｒ，ｓ］とが与えられていると仮定すると、$V_1$&$V_2$が、ＡＮＤ［２ｐ＋ｑ，２ｒ＋ｓ］として計算される。

図２１ｅには、３つのブール変数Ａ＝［ａ１，ａ２］、Ｂ＝［ｂ１，ｂ２］、Ｃ＝［ｃ１，ｃ２］の分割の結果が示されている。本発明者が選択した表現の興味深い側面は、同じブール式を計算するために使用可能な方法が幾つかあるということである。例えば、図２１ｅの文（３’）と文（４’）は、両方とも偽を変数に割り当てるが、互いに異って見える。同様に、文（５’）と文（６’）は互いに全く異っているが、両方ともＡ＆Ｂを計算する。

この変換の効力と弾力性とコストとの全てが、オリジナルの変数が分割される変数の個数に応じて増大する。弾力性は、実行時に符号化を選択することによってさらに増強される。言い換えれば、図２１ｂから図２１ｄまでの実行時ルック

アップテーブルは、コンパイル時には構築されないが（このことが、実行時ルックアップテーブルに対して静的解析を行うことを可能にするだろう）、混乱化アプリケーションに含まれるアルゴリズムによって構築される。当然のことながら、このことが、図２１ｅでの文（６’）で行われるように、基本演算を計算するためにインラインコードを使用することを防止する。

７．１．４　静的データの手続きデータへの転換

静的データ、特に文字列は、リバースエンジニアにとって有用な実際的情報を多く含んでいる。静的ストリングを混乱化するための方法は、静的ストリングをそのストリングを生成するプログラムの形に変換することである。ＤＦＡまたはトライ走査(Trie traversal)であることが可能であるプログラムが、他のストリングも生成することが可能である。

例えば、ストリング“ＡＡＡ”、“ＢＡＡＡＡ”、“ＣＣＢ”を混乱化するように構築されている図２２の関数Ｇを考察しよう。Ｇによって生成される値は、Ｇ（１）＝“ＡＡＡ”、Ｇ（２）＝“Ｂ

ＡＡＡＡ”、Ｇ（３）＝Ｇ（５）＝“ＣＣＢ”、および、（実際にはプログラムで使用されない）Ｇ（４）＝“ＸＣＢ”である。他の引数値の場合には、Ｇが終了しても終了しなくてもよい。

当然のことながら、全ての静的ストリングデータの計算を単一の関数の形に集合することは、極めて望ましくない。ソースプログラムの「通常の」制御流れの中に埋め込まれたより小さなコンポーネントの形にＧ関数が分割されている場合には、はるかに高度な効力と弾力性とが実現される。

この手法を節６．２．５のテーブル解釈変換と組み合わせることが可能であるということを指摘しておくことが重要である。その混乱化の意図は、Javaバイトコードの１つのセクションを別の仮想計算機のためのコードに変換するということである。この新たなコードが、典型的には、被混乱化プログラム内の静的ストリングデータとして記憶されるだろう。しかし、さらに高いレベルの効力と弾力性とを得るためには、上記ストリングが、上記のように、そのストリングを生成するプログラムに転換されることも可能である。

７．２　集合変換

命令型言語および機能言語とは対照的に、オブジェクト指向言語は、制御指向であるよりもデータ指向である。言い換えれば、オブジェクト指向プログラムでは、制御が、データ構造まわりに編成されるが、他の仕方では編成されない．このことは、オブジェクト指向アプリケーションのリバースエンジニアリングの重要部分は、プログラムのデータ構造の復元を試みることであることを意味する。逆に、混乱化器がこうしたデータ構造を隠蔽しようとすることが重要である。

オブジェクト指向言語の殆どでは、データの集合を行うための方法は２つだけであり、すなわち、配列の形でのデータ集合と、オブジェクトの形でのデータ集合である。次の３つのセクションでは、こうしたデータ構造が混乱化されることが可能な方法を検討する。

７．２．１　スカラ変数の併合

$V_1$，．．．，$V_k$の組合せ範囲が$V_M$の精度に適合するならば、２つ以上のスカラ変数$V_1$，．．．，$V_K$が、１つの変数$V_M$の形に併合されることが可能である。例えば、２つの３２ビット整変数が、１つの６４ビット変数に併合されることが可能である。個々の変数に対する演算が、ＶＭに対する演算の形に変換されるだろう。簡単な例として、２つの３２ビット整変数Ｘ、Ｙを６４ビット変数Ｚの形へ併合することを考察する。次の併合式

$$Z(X,Y)=2^{32}*Y+X$$

を使用して、図２３ａの算術恒等式が得られる。図２３ｂには幾つかの簡単な例が示されている。特に、図２３は、２つの３２ビット整変数Ｘ、Ｙを１つの６４ビット変数Ｚへ併合することを示している，。ＹがＺの上側３２ビットを占め、かつ、Ｘが下側３２ビットを占める。ＸまたはＹのどちらかの実際の値域がプログラムから演繹されることが可能である場合には、直感的により一層分かりにくい併合が使用されることも可能である。図２３ａは、ＸとＹによる加算と乗算とのための規則を示している。図２３ｂは、幾つかの簡単な例を示す。この例は、例えば（2’）と（3’）とを「Ｚ＋＝４７２４４６４０２６１」の形に併合する事によって、さらに混乱化されることが可能である。

変数併合の弾力性は極めて低い。ある特定の変数が実際には２つの併合変数から成るということを推定するためには、混乱解除器は、算術演算セットがその特定の変数に適用されていることを調べるだけでよい。個々の変数に対する妥当な演算のいずれにも対応することが不可能であるボーガス（ｂｏｇｕｓ）演算を導入することに

よって、弾力性を増大させることが可能である。図２３ｂの例では、例えば回転によって、すなわち、Ｉｆ（$P^{\Gamma}$）Ｚ－ｒｏｔａｔｅ（Ｚ，５）によって、Ｚの２つの半分部分を併合するように見える演算を挿入することが可能である。

この変換の１つの変形は、$V_1$，．．．，$V_K$を、次のような適切なタイプの１つの配列に併合することである。

$$V_A = \begin{matrix} 1 \ldots k \\ V_1 \ldots V_k \end{matrix}$$

$V_1$，．．．，$V_k$がオブジェクト参照変数である場合には、例えば、ＶＡの要素タイプが、継承階層において$V_1$，．．．，$V_k$のタイプのいずれよりも高いレベルにある全てのクラスであることが可能である。

７．２．２　配列の再構成

配列に対して行われる演算を混乱化するために、幾つかの変換が考案されることが可能である。例えば、１つの配列を幾つかの二次配列に分割するか、２つ以上の配列を１つの配列に併合するか、１つの配列を折り畳む（ｆｏｌｄ）（次元数を増大させる）か、または、１つの配列を平坦化する（ｆｌａｔｔｅｎ）（次元数を減少させる）ことが可能である。

図２４は、配列再構成の幾つかの例を示す。文（１－２）では、配列Ａが２つの二次配列Ａ１、Ａ２の形に分割される。Ａ１が、偶数のインデックスを有するＡの要素を保持し、Ａ２が、奇数のインデックスを有する要素を保持する。

図２４の文（３－４）が、どのようにして整数配列Ｂ、Ｃが結果としての配列ＢＣの形に交互配置されることが可能であるかを示している。配列Ｂからの要素と配列Ｃからの要素とが、その結果得られる配列全体にわたって均一に分散させられている。

文（６－７）は、１次元配列Ｄが２次元配列Ｄ１の形にどのように折り畳まれることが可能であるかを示している。最後に、文（８－９）は逆変換を示している。２次元配列Ｅが１次元配列Ｅ１の形に平坦化される。

配列の分割と折畳みとが$u_6$データ複雑性メトリックを増大させる。一方、配列の併合と平坦化とが、このメトリックを減少させる。このことは、これらの変換が僅かなまたはネガティブな効力だけしか持たないということを示しているように見えるかも知れないが、実際には、これは誤りを生じさせやすい。問題は、幾つかのデータ構造変換の重要な側面を把握することには、図７の複雑性メトリックが役立たないということである。すなわち、こうした変換が、当初は存在しなかった構造を導入するか、または、オリジナルのプログラムから構造を取り除くことになる。このことは、プログラムの混乱化を著しく増大させることが可能である。例えば、２次元配列を宣言するプログラマは、意図的にそうするのである。選択された構造が、操作されているデータをとにかく適切にマップする。その配列が１次元構造に折り畳まれる場合には、リバースエンジニアは、貴重な実際的情報を奪われてしまっていることになるだろう。

７．２．３　継承関係の修正

Java™言語のような現在のオブジェクト指向言語では、主要なモジュール化および抽象化概念はクラスである。クラスは、データ（インスタンス変数）と制御（方法）とをカプセル化する本質的に抽象的なデータタイプである。クラスはＣ－（Ｖ，Ｍ）と記述され、前式のＶがＣのインスタンス変数のセットであり、Ｍがその方法である。

抽象データタイプの従来の概念とは対照的に、２つのクラス$C_1$、$C_2$が、集合化（$C_2$がタイプ$C_1$のインスタンス変数を有する

）と継承（新たな方法とインスタンス変数とを加えることによって、$C_2$が$C_1$を拡張する）とによって構築されることが可能である。継承は$C_2 = C_1 \cup C'_2$と記述される。$C_2$は、そのスーパークラスまたは親クラスである$C_1$を継承すると表現される。Ｕ演算子は、$C'_2$で定義される新たなプロパティと親クラスとを組み合わせる関数である。Ｕの正確なセマンティクスは、個々のプログラミング

言語に依存している。Javaのような言語では、一般的に、Uが、インスタンス変数に適用される場合には合併として解釈され、一方、方法に適用される場合にはオーバーライドと解釈される。

メトリック$u_7$にしたがって、クラス$C_1$の複雑性が、継承階層と直接の子孫の個数とにおけるその深さ（ルートからの距離）に応じて増大する。例えば、この複雑性を増加させることが可能な方法が２つある。すなわち、図２５ａに示されているように、ある１つのクラスを分割（ファクタ）することと、図２５ｂに示されているように、新たな、にせ(bogus)の、クラスを挿入することとが可能である。

クラスファクタリングに関する問題点はその弾力性である。ファクタされたクラスを混乱解除器が簡単に併合することを阻止するものは全くない。これを防止するために、一般的には、図２５ｄに示されているように、ファクタリングと挿入とが組み合わされる。これらのタイプの変換の弾力性を増加させる別の方法は、導入されたクラス全てに関して新たなオブジェクトが生成されることを確実なものにすることである。

図２５ｃは、「偽のリファクタリング」と呼ばれる、クラス挿入の変形を示している。リファクタリングは、その構造が劣化してしまっているオブジェクト指向プログラムを再構築するための（時として自動の）手法である（本明細書に参考として採り入れられてい

る、William F. Opdyke and RalphE. Johnson. Creating abstract superclasses by refactoring. In Stan C.Kwansny and John F. Buck, editors, Proceedings of the 21st Annual Conference on Computer Science, page 66-73, New York, NY, USA, February 1993. ACM Press. ftp://st. cs.uiuc.edu/pub/papers/refactoring/refactoring-superclasses.psを参照されたい）。リファクトリングは２ステップのプロセスである。第１に、見掛け上は別個の２つのクラスが事実上は同様の動作をするということが検出される。その次に、両方のクラスに共通する特徴が、新たな（おそらくは抽象的な）親クラスの中に移される。偽のリファクタリングは同様の操作であるが、共通の動作を持たない２つのクラス$C_1$

、$C_2$に対してだけ行われる。両方のクラスが同じタイプのインスタンス係数を有する場合には、これらのクラスが新たな親クラス$C_3$の中に移される。$C'_3$の方法が、$C_1$と$C_2$からの方法のバグだらけ（ｂｕｇｇｙ）なバージョンであり得る。

７．３　オーダリング変換

セクション６．４では、（可能な場合にではあるが）計算が行われる順序をランダム化することが有用な混乱化であることを示した。同様に、ソースアプリケーションにおける宣言の順序をランダム化することが有益である。

特に、本発明においては、クラス内の方法およびインスタンス変数と方法の仮パラメタ方法との順序をランダム化する。後者の場合には、当然のことながら、対応する実際の順序が再オーダリングされなければならない。こうした変換の効力は低く、弾力性は片方向である。

多くの場合には、配列中の要素を再オーダリングすることも可能だろう。簡単に述べると、本発明では、オリジナルの配列内のｉ番目の要素を、再オーダリングされた配列の新たな位置にマップする、不明瞭性（ｏｐａｑｕｅ）符号化関数ｆ（ｉ）が提供される。

```
 {                                        {
int i=1, A[1000];                        int i=1, A[1000];
while (i < 1000)          = T =>         while (i < 1000)
     ...A[i]...;                              ...A[f(i)]...;
     i++;                                     i++;
}                                         }
```

８．不明瞭値と不明瞭述語

上記のように、不明瞭な述語が、制御流れを混乱化する変換の設計における主要なビルディングブロックである。事実として、殆どの制御変換の品質が、こうした述語の品質に直接的に依存している。

セクション６．１では、トリビアルで弱い弾力性を有する単純な不明瞭述語の例を示した。これは、ローカル静的分析またはグローバル静的解析を使用して不

明瞭述語が解読されることが可能である（自動混乱解除器がその値を発見することができる）ということを意味する。当然のことながら、一般的に、攻撃に対するはるかにより高度の抵抗性が必要とされている。理想的には、それを解読するのには（プログラムのサイズにおいて）最悪指数関数的時間を要するが、それを構築するには多項式的時間しか要さない不明瞭述語を、構築することが可能であることが望ましい。このセクションでは、２つのこうした手法を説明する。第１の手法はエイリアシングに基づいており、第２の手法は軽量プロセス（ｌｉｇｈｔｗｅｉｇｈｔ　ｐｏｅｓｓ）に基づいている。

８．１　オブジェクトとエイリアスとを使用する不明瞭構造体

エイリアシングが可能である場合には常に、手続き間静的分析が著しく複雑化される。実際に、動的割付けとループとＩＦ文を有する言語においては、正確で流れ依存形のエイリアス解析は決定不可能である。

このセクションでは、低コストでかつ自動混乱解除攻撃に対して弾力性がある不明瞭述語を構築するために、エイリアス解析の困難さが利用される。

８．２　スレッドを使用する不明瞭構造体

並列プログラムは順次プログラムに比べて静的分析を行うこと困難である。その理由は、並列プログラムのインターリービングセマンティクスである。すなわち、並列領域ＰＡＲ　$S_1$，$S_2$，．．．，$S_n$，ＥＮＤＰＡＲ内のｎ個の文が、ｎ！個の異った方法で実行されることが可能である。これにも係わらず、並列プログラムの幾つかの静的解析が、多項式的時間［１８］で行われることが可能であり、一方、他は、ｎ！個のインタリービング全てが考慮されることを必要とする。

Ｊａｖａでは、並列領域が、スレッドとして知られている軽量プロセスを使用して構築される。（本発明者の視点から見て）Ｊａｖａスレッドは２つの有益な特性を有する。すなわち、（１）Ｊａｖａスレッドのスケジューリングポリシーは言語仕様によっては厳密に指定されておらず、したがって、インプリメンテーションに依存することになり、（２）スレッドの実際のスケジューリングが、ユーザインタラクションによって生成される非同期イベントのような非同期イベン

トと、ネットワークのトラフィックとに依存することになるという特性を有する。並列領域の固有インタリービングセマンティクスと組み合わされる時には、このことは、スレッドを静的解析することが非常に困難であるということを意味する。

本発明では、解読には最悪指数関数的時間を必要とする不明瞭述語（図３２を参照されたい）を生成するために、上記の考察結果が利用される。この基本的な着想は、セクション８．２で使用されている着想と非常に類似している。すなわち、グローバルデータ構造Ｖが生成され、時たま更新されるが、不明瞭問合せが行われることが可能であるような状態に維持される。相違点は、現在実行中のスレッドによってＶが更新されるということである。

当然のことながら、Ｖは、図２６で生成された動的データ構造のような動的データ構造であることが可能である。スレッドが、移動と挿入のための呼出しを非同期的に実行することによって、そのスレッドの個々のコンポーネント内においてグローバルポインタｇ、ｈをランダムに移動させるだろう。これは、非常に高い弾力性を得るために、データ競合をインタリービング効果およびエイリアシング効果と組み合わせるという利点を有する。

図２７では、Ｖが１対のグローバル整変数Ｘ、Ｙである、よりはるかに単純な例を使用して、上記の着想を図解している。これは、任意の整数ｘおよび整数ｙの場合に「$7y^2-1$」が$x^2$に等しくないという基本数論からの公知の事実に基づいている。

９．混乱解除と予防変換

本発明者の混乱化変換の多く（特にセクション６．２の制御変換）は、実プログラム内にボーガスプログラム（ｂｏｇｕｓ　ｐｒｏｇｒａｍ）を埋め込むと言い表されることが可能である。言い換えれば、被混乱化アプリケーションは、実際には、１つに併合された２つのプログラムから成り、すなわち、有用なタスクを行う実プログラムと、無益な情報を計算するボーガスプログラムとが、１つに併合されている。このボーガスプログラムの唯一の目的は、無関係

なコードの背後に実プログラムを隠蔽することによって、潜在的なリバースエンジニアを混乱させることである。

上記不明瞭述語は、ボーガス内部プログラムが容易に識別されて除去されることを防止するために、混乱化器が自由に使用できる主要な仕掛けである。例えば、図２８ａでは、混乱化器が、実プログラムの３つの文の中に、不明瞭述語によって保護されたボーガスコード（ｂｏｇｕｓ　ｃｏｄｅ）を埋め込む。混乱解除器のタスクは、被混乱化アプリケーションを調べて、内部ボーガスコードを自動的に識別して取り除くことである。これを行うために、混乱解除器が、最初に不明瞭構造体を識別してから、その構造体を評価しなければならない。このプロセスが図２８ｂから図２８ｄに示されている。

図２９は、半自動的混乱解除ツールの構造を示す。このツールは、リバースエンジニアリングのコミュニティでは公知である幾つかの手法を採り入れている。このセクションの残り部分では、こうした手法の幾つかを簡単に検討し、混乱解除をより困難にするために混乱化器を使用することが可能である様々な対抗策（いわゆる予防変換）を説明する。

９．１　予防変換

図２ｇに関連して上記で説明した予防変換は、制御変換またはデータ変換とは趣が全く異っている。制御変換またはデータ変換とは対照的に、予防変換の主要な目標は、人間の読み手に対してプログラムを覆い隠すことではない。むしろ、予防変換は、公知の自動混乱解除手法をより困難にするように（本来予防変換）、または、現在の混乱解除器またはデコンパイラにおける既知の問題を探り出すように（目標予防変換）設計される。

９．１．１　本来予防変換

本来予防変換は、一般的に、低い効力と高い弾力性とを有する。最も重要なことは、本来予防変換が他の変換の弾力性を増強する能力を有するだろうということである。一例として、セクション６．４で示唆されているように、ｆｏｒループを逆方向に実行するように再オーダリングし終わっているものと仮定する。ループがループ運搬データ従属性を持たないことを調べることが可能だったという

理由だけから、この変換が適用されることが可能だった。当然のことながら、混乱解除器が同じ解析を行ってループを順方向実行に戻すことを阻止するものは何もない。これを防止するために、逆ループに対するボーガスデータ従属性を加えることが可能である。

```
 {                                  {
for(i=1;i<=10;i++)   =T=>              int B[50];
     A[i]=i                            for(i=10;i<=1;i--)
}                                      A[i]=i;
                                       B[i]+=B[i*i/2]
                                   }
```

この本来予防変換がループ再オーダリング変換に加える弾力性は、ボーガス従属性の複雑性と従属性解析の技術的現状とに依存している［３６］。

９．１．２　目標予防変換

目標予防変換の一例として、ＨｏｓｅＭｏｃｈａプログラムを考察する（Ｍａｒｋ　Ｄ．ＬａＤｕｅ．ＨｏｓｅＭｏｃｈａ．ｈｔｔｐ：／／ｗｗｗ．ｘｙｎｙｘ．ｄｅｍｏｎ．ｎｌ／ｊａｖａ／ＨｏｓｅＭｏｃｈａ．ｊａｖａ，Ｊａｎｕａｒｙ　１９９７）。このプログラムは、Ｍｏｃｈａデコンパイラ（Ｈａｎｓ　Ｐｅｔｅｒ　Ｖａｎ　Ｖｌｉｅｔ．Ｍｏｃｈａ－－－Ｔｈｅ　Ｊａｖａ　ｄｅｃｏｍｐｉｌｅｒ．ｈｔｔｐ：／／ｗｅｂ．ｉｎｔｅｒ．ｎｌ．ｎｅｔ／ｕｓｅｒｓ／Ｈ．Ｐ．ｖａｎ．Ｖｌｉｅｔ／ｍｏｃｈａ．ｈｔｍｌ，Ｊａｎｕａｒｙ　１９９６）の弱点を調査するために特別に設計されている。ＨｏｓｅＭｏｃｈａプログラムは、ソースプログラム内の全ての方法の中のあらゆる復帰文の後に、特別な命令を挿入する。この変換は、アプリケーションの動作に対して全く影響を及ぼさないが、Ｍｏｃｈａをクラッシュさせるには十分である。

９．２　不明瞭構造体の識別と評価

混乱解除の最も重要な部分は、不明瞭構造体の識別と評価である。識別と評価が別個のアクティビティであることに留意されたい。不明瞭構造体は、ローカルである（単一の基本ブロック内に含まれる）か、または、グローバルである（単

一の手続きに含まれる）か、または、手続き間である（プログラム全体にわたって分散している）ことが可能である。例えば、ｉｆ（ｘ＊ｘ＝＝（$7^{F}$＊ｙ＊ｙ－１））がローカル不明瞭述語であり、Ｒ＝Ｘ＊Ｘ；．．．；Ｓ＝７＊ｙ＊ｙ－１；．．．；ｉｆ（Ｒ＝＝$S^{r}$）．．．がグローバル不明瞭術語である。ＲとＳの計算が互いに異った手続きで行われた時には、構造体は手続き間不明瞭構造体であるだろう。当然のことながら、ローカル不明瞭述語の識別は、手続き間の不明瞭述語の識別よりも容易である。

９．３　パターン照合による識別

混乱解除器は、不明瞭述語を識別するために、既知の混乱化器によって使用されるストラレジの知識を使用することが可能である。混乱解除器の設計者は、（混乱化器をデコンパイルすることによって、または、単純に、混乱化器が生成する被混乱化コードを調査することによって）混乱化器を調査することが可能であり、一般的に使用される不明瞭述語を識別することが可能なパターン照合規則を構築することが可能である。この方法は、ｘ＊ｘ＝＝（７＊ｙ＊ｙ

－１）またはｒａｎｄｏｍ（$1^{F}$，５）＜０のような単純なローカル述語に対して最も効果的に働くだろう。

パターン照合の試みを妨害するために、混乱化器は、常套的な不明瞭構造体の使用を避けなければならない。さらに、実アプリケーションで使用される構造体に構文上類似している不明瞭構造体を選択することも重要である。

９．４　プログラムスライシングによる識別

プログラマは、一般的に、プログラムの被混乱化バージョンが、リバースエンジニアにとってオリジナルのプログラムよりも理解が困難であることを発見するだろう。その主たる理由は、被混乱化プログラムでは、（ａ）生きている「実」コードに、死んだボーガスコードがちりばめられ、かつ、（ｂ）論理的に関係付けられたコード断片が分解されてプログラム全体にわたって分散されるということである。プログラムスライシングツールが、こうした混乱化に対抗するためにリバールエンジニアによって使用されることが可能である。こうしたツールは、スライスと呼ばれる管理可能なチャンク（ｃｈｕｎｃｋ）の形にプログラムを分

解するために、リバースエンジニアを対話式に補助することが可能である。ポイントｐと変数ｖとに関するプログラムＰのスライスは、ｐにおいてｖの値に寄与することが可能だったＰの文の全てから成る。したがって、プログラムスライサは、混乱化器がこうした文をプログラム全体に分散させた場合にさえ、不明瞭変数ｖを計算するアルゴリズムの文を、被混乱化プログラムから抽出することが可能だろう。

スライシングを有効性がより劣る識別ツールにするために混乱化器で使用可能な幾つかのストラテジが存在する。Ａｄｄパラメタエイリアスパラメタエイリアスは、同じ記憶場所を参照する２つの仮パラメタ（または、仮パラメタとグローバル変数）である。厳密

な手続き間スライシングは、プログラム中の潜在的エイリアスの個数に応じて増大し、一方、この潜在的エイリアスの個数は、仮パラメタの個数に応じて指数関数的に増大する。したがって、混乱化器が、エイリアシングされたダミーパラメタをプログラムに追加する場合には、その混乱化器が、（厳密なスライスが要求される場合には）スライサを著しく減速させるか、または、（高速スライシングが必要とされる場合には）スライサに非厳密なスライスを生じさせるように強制する。

Ｕｎｒａｖｅｌ（Ｊａｍｅｓ Ｒ．Ｌｙｌｅ，Ｄｏｌｏｒｒｅｓ Ｒ．Ｗａｌｌａｃｅ，Ｊａｍｅｓ Ｒ．Ｇｒａｈａｍ，Ｋｅｌｔｈ Ｂ．Ｇａｌｌａｇｈｅｒ，Ｊｏｓｅｐｈ Ｐ．Ｐｏｏｌｅ，ａｎｄ Ｄａｖｉｄ Ｗ． Ｂｉｎｋｌｅｙ．Ｕｎｒａｖｅｌ：Ａ ＣＡＳＥ ｔｏｏｌ ｔｏ ａｓｓｉｓｔ ｅｖａｌｕａｔｉｏｎ ｏｆ ｈｉｇｈ ｉｎｔｅｇｒｉｔｙ ｓｏｆｔｗａｒｅ．Ｖｏｌｕｍｅ １：Ｒｅｑｕｉｒｅｍｅｎｔｓ ａｎｄ ｄｅｓｉｇｎ．Ｔｅｃｈｎｉｃａｌ Ｒｅｐｏｒｔ ＮＩＳ－ＴＩＲ ５６９１，Ｕ．Ｓ．Ｄｅｐａｒｔｍｅｎｔ ｏｆ Ｃｏｍｍｅｒｃｅ，Ａｕｇｕｓｔ １９９５）のような一般に普及しているスライシングツールのような加算変数従属性（ａｄｄ ｖａｒｉａｂｌｅ ｄｅｐｅｎｄｅｎｃｙ）は、小さなスライスの計算には適切に機能するが、より大きいスライスの計算に対しては過大な時間を要する場合がある。例えば

、４０００行のCプログラムに対して使用した場合には、Ｕｎｒａｖｅｌがスライス計算を完了するのに３０分間を越える時間を必要としたケースがあった。こうした特徴を強制的に引き出すために、混乱化器が、ボーガス変数従属性を加えることによってスライスサイズを増大させようとしなければならない。次の例では、見掛け上ではｘの値に寄与するが実際には寄与しない２つの文を

加えることによって、スライス計算のサイズｘが増大させられている。

```
main()          {          =T =>          main()  {
int x=1;                                  int x=1;
x = x * 3;                                     if (PF ) x++;
}                                         x = x + V0 ;
                                          x = x * 3;
                                          }
```

９．５　静的解析

混乱解除器は、述語全ての結果を分析するために、被混乱化プログラムを利用することが可能である。このようにして被混乱化アプリケーションの実行時の特徴を調査する任意の混乱解除方法、すなわち、静的解析が呼び出されるだろう。混乱解除器は、多数のテストランにおいて常に同じ真理値を戻す述語が不明瞭$P^T$（$P^F$）述語であることが判明するかも知れないので、こうした真理値を戻す述語の全てに対するリバースエンジニアの注意を促す。混乱解除器は、こうした述語を真（偽）でむやみに置き換えることは不可能であるが、これは、そうすることが危険過ぎる可能性があるからである。多くのアプリケーションは、例外的な状況下でしか発生しない条件を検査する「真の」述語を含むが、混乱解除器にとっては、こうしたアプリケーションは、不明瞭述語と同じに動作するように見えるだろう。例えば、ｐｉｆ（Ｌｅａｐ　Ｙｅａｒ）．．．を考察されたい。

静的解析は、さらに、評価のためにも使用されることが可能である。プログラムM内の潜在的な不明瞭述語（例えば、$P^T$）がすでに識別されている時に、その値（真）が推定され、上記不明瞭述語がこの推定値ですでに置き換えられている被混乱化プログラムのバ

ージョンM’が作成される。その次に、同一の入力に対してMとM’とが並行に実行され、MとM’とが同一の出力を生じさせることを確認するために比較が行われる。これらの出力が同一である場合には、図３０に示されているように、その述語が、真のアプリケーションではないボーガスの一部であったと結論付けられることが可能である。

選択された入力が上記プログラム中の全てのパスを適切にカバーすることが確保されなければならない。同様に、上記プログラムが、稀にしかとられないパス（ｉｆ（Ｌｅａｐ　Ｙｅａｒ）．．．）を含む場合には、これは困難だろう。さらに、多数の正しい入力／出力データを生成することは非常に困難であり、特にアプリケーションの内部構造が未知である時、または、（往々にしてＪａｖａプログラムに当てはまるように）複雑なグラフィカルユーザインタフェースを通して入力が行われる時には、これは非常に困難である。

静的解析による識別を防止するために、混乱化器が、$P^T$述語または$P^F$述語を挿入する変換よりも、（図１１ｂに示されているような）$P^?$述語を挿入する変換を優先するように選択してもよい。

静的解析に対する別の採用可能な対抗策は、幾つかの述語が同時に解読されなければならないように、不明瞭述語を設計することである。これを行うための方法の１つは、不明瞭述語に副作用を持たせることである。下記の例では、混乱化器が、（一種の静的流れ解析によって）、文$S_1$、$S_2$が常に同じ回数だけ実行しなければならないということを決定する。これらの文が、関数$Q_1$と関数$Q_2$とに対する呼出しである不明瞭述語を導入することによって混乱化される。関数$Q_1$と関数$Q_2$はグローバル変数を増減する。

```
{                    =T  =>        {
S1;                                   int k=0;
```

```
S2;                          bool Q1 (x)     {
}                            k+=2^j1 ; return (P^T 1)}
                             bool Q2 (x)     {
                             k-=2^j1 ; return (P^T 2)}
                            {
                             if (Q1 (j) T ) S1 ;
                                  . . .
                             if (Q2 (k) T ) S2 ;
                             }
```

混乱解除器が１つの（両方ではない）述語を真で置き換えようとする場合には、ｋがオーバフローするだろう。その結果として、混乱解除されたプログラムが、エラー状態で終了するだろう。

９．６　データフロー解析による評価

混乱解除は、様々なタイプのコード最適化に類似している。ｉｆ（Ｆａｌｓｅ）．．．を取り除くことは、死んだコード（ｄｅａｄ　ｃｏｄｅ）削除であり、ｉｆ文ブランチ（例えば、図２８における$S_1$と$S_0^1$）から同一コードを移動することは、コードホイスティング（ｃｏｄｅ　ｈｏｉｓｔｉｎｇ）であり、これらは一般的なコード最適化手法である。

不明瞭構造体が識別され終わると、その構造体の評価を試みることが可能になる。単純な事例では、到達定義データフロー解析（ｒｅａｃｈｉｎｇ　ｄｅｆｉｎｉｔｉｏｎ　ｄａｔａ－ｆｌｏｗ　ａｎａｌｙｓｉｓ）を使用する定数伝搬で十分であることが可能である。ｘ＝５；．．．；ｙ＝７；．．．；ｉｆ（ｘ＊ｘ＝＝（７＊ｙ＊ｙ＝１）．．．

９．７　定理の証明による評価

データフロー解析が不明瞭述語を解読するのに十分なだけ強力ではない場合には、混乱解除器が定理の証明を使用することを試みることが可能である。これが可能であるか不可能であるかは、（確認が困難な）技術的現状の定理証明プログラムの能力と、証明されることが必要な定理の複雑性とに依存して

いる。当然のことながら、帰納によって証明可能な定理（例えば、$x^2$（x＋1）$^2$＝0（ｍｏｄ　4））は、十分に現行の定理証明プログラムの到達範囲内である。

事柄をさらに困難にするために、証明が困難であることが知られている定理、または、それに関する既知の証明が存在しない定理を使用することが可能である。下記の例では、混乱解除器が、$S_2$が生きたコード（ｌｉｖｅ　ｃｏｄｅ）であることを調査するために、ボーガスループ（ｂｏｇｕｓ　ｌｏｏｐ）が常に終了するということを証明しなければならないだろう。

```
  {                    =T =>          {
S1;                                   S1;
S2;                                   n = random(1,2^32);
}                                     do
                                            n = ((n%2)!=0)?3*n+1:n/2
                                      while (n>1);
                                      S2;
                                       }
```

これは、Ｃｏｌｌａｔｚ問題として知られている。上記ループが常に終了するだろうということが推測される。この推測の既知の根拠は存在しないが、７＊１０$^{11}$までの全ての数に関してそのコードが終了することが知られている。したがって、この混乱化は安全であり（オリジナルの混乱化されたコードが全く同じに動作する）、混乱解除を行うことは困難である。

## ９．８　混乱解除および部分評価

混乱解除は、さらに、部分評価にも類似している。部分評価器は、プログラムを、２つの部分、すなわち、部分評価器によって事前計算されることが可能である静的部分と、実行時に実行される動的部分とに分割する。動的部分は、混乱化されていないオリジナルのプログラムに相当するだろう。静的部分は、ボーガス内部プログラムに相当し、このボーガス内部プログラムは、識別された場合には、混乱解除時点で評価され除去されることが可能である。

他の静的内部手続き解析手法の全てと同様に、部分評価はエイリアシングの影響を受けやすい。したがって、スライシングに関連して言及した予防変換と同じ予防変換が、部分評価にも適用される。

１０．混乱化アルゴリズム

次に、セクション３の混乱化器アーキテクチャと、セクション５の混乱化品質の定義と、セクション６からセクション９の様々な混乱化変換の説明とに基づいて、本発明の実施様態の１つによるさらに詳細なアルゴリズムを説明する。

混乱化ツールの最上位レベルのループは、次の一般構造を有することが可能である。

```
WHILE NOT Done(A)DO
      S:=SelectCode(A);
      T:=SelectTransform(S);
      A:=Apply(T,S);
END;
```

ＳｅｌｅｃｔＣｏｄｅは、混乱化されるべきその次のソースコードオブジェクトを戻す。ＳｅｌｅｃｔＴｒａｎｓｆｏｒｍは、この特定のソースコードオブジェクトを混乱化するために使用されなければならない変換を戻す。Ａｐｐｌｙが、変換をソースコードオブジェクトに適用し、それに応じてアプリケーションを更新する。Ｄｏｎｅは、所要レベルの混乱化に達し終わった時を決定する。これらの関数の複雑性は、混乱化ツールのソフィスティケーションに依存するだろう。単純な位取りの結果として、ＳｅｌｅｃｔＣｏｄｅとＳｅｌｅｃｔＴｒａｎｓｆｏｒｍとが単純にランダムなソースコードオブジェクト／変換を戻し、Ｄｏｎｅが、特定の限界をアプリケーションのサイズが越える時にループを終了させることが可能である。通常では、こうした動作は不十分である。

アルゴリズム１は、さらにはるかに洗練された選択動作および終了動作を含むコード混乱化ツールの記述を与える。実施様態の１つでは、このアルゴリズムが幾つかのデータ構造を使用し、こうしたデータ構造がアルゴリズム５、６、７に

よって構築される。

各ソースコードオブジェクトＳに関するＰ$_S$である時に、Ｐ$_S$（Ｓ）は、プログラマがＳで使用した言語構造体セットである。Ｐ$_S$（Ｓ）が、Ｓに関する適切な混乱化変換を発見するために使用される。

各ソースコードオブジェクトＳに関するＡである時に、Ａ（Ｓ）＝｛Ｔ$_i$－－＞Ｖ$_1$；．．．；Ｔ$_n$＝－＞Ｖ$_n$｝は、変換Ｔ$_i$から値Ｖ$_i$へのマッピングであり、Ｔ$_i$をＳに適用することがどのように適切であるかを記述する。この着想は、Ｓにとって「不自然である」新たなコードを特定の変換が導入するので、こうした変換が特定のソースコードオブジェクトＳに関して不適切である可能性があるということであるこの新たなコードは、Ｓの中では場違いに見え、したがって、リバースエンジニアには発見しやすいだろう。適切性値（ａｐｐｒｏｐｒｉａｔｅｎｅｓｓ　ｖａｌｕｅ）Ｖ$_i$が大きければ大きいほど、変換Ｔ$_i$によって導入されるコードがよ

り良好に適合するだろう。

各ソースコードオブジェクトＳに関するＩである時に、Ｉ（Ｓ）はＳの混乱化プライオリティである。Ｉ（Ｓ）が、Ｓの内容を混乱化することがどれだけ重要であるかを記述する。Ｓが重要なトレードシークレットを含む場合には、Ｉ（Ｓ）がＨＩＧＨであるだろうし、一方、それが主として「ブレッドアンドバター」コードを含む場合には、Ｉ（Ｓ）がＬＯＷだろう。

各ルーチンＭに関するＲである時に、Ｒ（Ｍ）はＭの実行時間ランクである。他のいずれのルーチンよりも多くの時間がＭを実行するために費やされる場合には、Ｒ（Ｍ）＝１である。

アルゴリズム１に対する一次入力は、アプリケーションＡと混乱化変換のセット｛Ｔ$_1$；Ｔ$_2$；．．．｝である。このアルゴリズムは、さらに、各々の変換に関する情報も要求し、特に、（セクション５での同名の関数と同様であるが、数値を戻す）３つの品質関数Ｔ$_{res}$（Ｓ）、Ｔ$_{pot}$（Ｓ）、Ｔ$_{cost}$（Ｓ）と、関数Ｐ$_t$を要求する。

ソースコードオブジェクトＳに適用される時に、Ｔ$_{res}$（Ｓ）が、変換Ｔの弾

力性（すなわち、Tが自動混乱解除器からの攻撃にどれだけ適切に耐えるか）の測度を戻す。

ソースコードオブジェクトSに適用される時に、$T_{pot}$（S）が、変換Tの効力（すなわち、Tによって混乱化された後に、人間がどれほど難しいSを理解するのか）の測度を戻す。

$T_{cost}$（S）が、TによってSに加えられる実行時間および空間ペナルティーの測度を戻す。

$P_t$が、Tがアプリケーションに加えることになる言語構造体のセットに各々の変換Tをマップする。

アルゴリズム1のポイント1からポイント3が、混乱化されるべきアプリケーションをロードし、適切な内部データ構造を構築する。ポイント4が、$P_S$（S）、A（S）、I（S）、R（M）を構築する。所要の混乱化レベルに達し終わるまで、または、最大実行時間ペナルティが越えられるまで、ポイント5が混乱化変換を適用する。最後に、ポイント6が新たなアプリケーションA'を書き換える。

アルゴリズム1（コード混乱化）

入力：

a）ソースコードまたはオブジェクトコードファイルで構成されているアプリケーションA　C1；C2；．．．。

b）言語によって定義される標準ライブラリL1；L2；．．．。

c）混乱化変換セット｛T1；T2；．．．｝。

d）変換Tの各々に関して、Tがアプリケーションに加えることになる言語構造体セットを与えるマッピングPt。

e）ソースコードオブジェクトSに対する変換Tの品質を表現する3つの関数$T_{res}$（S）、$T_{pot}$（S）、$T_{cost}$（S）。

f）Aに対する入力データセットI＝｛I1；I2；．．．｝。

g）2つの数値AcceptCost＞0およびReqObf＞0。AcceptCostが、ユーザが受け入れることになる最大の追加実行時間／空間ペナ

ルティの測度である。ＲｅｑＯｂｆが、ユーザによって要求される混乱化の量の測度である。
出力：ソースコードまたはオブジェクトコードファイルで構成されている被混乱化アプリケーションＡ’。
１．混乱化されるべきアプリケーション$C_1$；$C_2$；．．．をロードする。混乱化器は、
（ａ）ソースコードファイルをロードすることが可能であり、こ

の場合には混乱化器が、字句解析と構文解析と意味解析とを行う完全なコンパイラフロントエンドを含まなければならないだろうし（純粋に構文変換だけに自己限定する非力な混乱化器は、意味解析なしに処理することが可能である）、または、
（ｂ）オブジェクトコードファイルをロードすることが可能であり、オブジェクトコードがソースコード内に情報の大半または全てを保持する場合には（Ｊａｖａクラスファイルの場合のように）この方法が好ましい。
２．アプリケーションによって直接的または間接的に参照されるライブラリコードファイルＬ１；Ｌ２；．．．をロードする。
３．アプリケーションの内部表現を構築する。内部表現の選択は、混乱化器が使用するソース言語の構造と変換の複雑性とに依存する。典型的なデータ構造セットは、次のものを含んでよい。
（ａ）Ａ内の各ルーチンに関する制御流れグラフ。
（ｂ）Ａ内のルーチンに関する呼出しグラフ。
（ｃ）Ａ内のクラスに関する継承グラフ。
４．マッピングＲ（Ｍ）および$P_s$（Ｓ）（アルゴリズム５を使用）、Ｉ（Ｓ）（アルゴリズム６を使用）、及びＡ（Ｓ）（アルゴリズム７を使用）を構築する。
５．アプリケーションに混乱化変換を適用する。各ステップにおいて、混乱化されるべきソースコードオブジェクトＳと、Ｓに対して適用すべき適切な変換Ｔとが選択される。所要の混乱化レベルに達した時に、または、許容可能な実行時間

コストを超過した時に、このプロセスが終了する。
ＲＥＰＥＡＴ
　Ｓ　：＝ＳｅｌｅｃｔＣｏｄｅ（Ｉ）；
　Ｔ　：＝ＳｅｌｅｃｔＴｒａｎｓｆｏｒｍ（Ｓ，Ａ）；

　ＴをＳに適用し、ポイント３からの適切なデータ構造を更新する；
ＵＮＴＩＬ　Ｄｏｎｅ（ＲｅｑＯｂｆ，ＡｃｃｅｐｔＣｏｓｔ，Ｓ，Ｔ，Ｉ）．
６．被混乱化ソースコードオブジェクトを新たな被混乱化アプリケーションＡ’に再構成する。
アルゴリズム２（ＳｅｌｅｃｔＣｏｄｅ）
入力：アルゴリズム６によって計算される通りの混乱化プライオリティマッピング。
出力：ソースコードオブジェクトＳ。
　Ｉが各々のソースコードオブジェクトＳをＩ（Ｓ）にマップし、このＩ（Ｓ）は、Ｓを混乱化することがどれだけ重要であるかの測度である。混乱化すべき次のソースコードオブジェクトを選択するために、Ｉが優先待ち行列として処理されることが可能である。言い換えれば、Ｉ（Ｓ）が最大化されるようにＳが選択される。
アルゴリズム３（ＳｅｌｅｃｔＴｒａｎｓｆｏｒｍ）
入力：
　ａ）ソースコードオブジェクトＳ。
　ｂ）アルゴリズム７によって計算される通りの適切性マッピングＡ。
出力：変換Ｔ
　ある特定のソースコードオブジェクトＳに適用するための最も適切な変換を選択するために、任意の個数の発見法（ｈｅｕｒｉｓｔｉｃｓ）が使用可能である。しかし、考慮すべき２つの重要な問題がある。第１に、選択された変換が、Ｓ内のコードの残り部分と自然に混じり合わなければならない。これは、Ａ（Ｓ）において高い適切性値を有する変換を優先することによって取扱可能である。第

２に、高い「出費に見合うだけの価値（ｂａｎｇ－ｆｏｒ－ｔｈｅ－ｂｕｃｋ）」を得る（すなわち、低い実行時間ペナルティで高レベルの混乱化を得る）変換が優先されることが望ましい。これは、効力と弾力性を最大化すると同時にコストを最小化する変換を選択することによって実現される。これらの発見法が次のコードによって取り込まれ、このコード内のｗ１、ｗ２、ｗ３がインプリメンテーション定義定数である。

Ｔ－－＞ＶがＡ（Ｓ）内であるように変換Ｔを戻し、かつ、（ｗ１＊$T_{pot}$ｔ（Ｓ）＋ｗ２＊$T_{res}$（Ｓ）＋Ｗ３＊Ｖ）／$T_{cost}$（Ｓ）が最大化される。

アルゴリズム４（Ｄｏｎｅ）

入力

ａ）ＲｅｑＯｂｆ、混乱化の残留レベル。

ｂ）ＡｃｃｅｐｔＣｏｓｔ、残留する許容可能な実行時間ペナルティ。

ｃ）ソースコードオブジェクトＳ。

ｄ）変換Ｔ。

ｅ）混乱化プライオリティマッピングＩ。

出力：

ａ）更新されたＲｅｑＯｂｆ。

ｂ）更新されたＡｃｃｅｐｔＣｏｓｔ。

ｃ）更新された混乱化プライオリティマッピングＩ。

ｄ）終了条件に達している場合には真であるブール返却値。

Ｄｏｎｅ関数が２つの目的を満たす。この関数は、ソースコードオブジェクトＳが混乱化され終わっており、かつ、縮小された優先順位値を受け取らなければならないという事実を反映させるために、優先待ち行列Ｉを更新する。この縮小は、変換の弾力性と効力と

の組合せに基づいている。Ｄｏｎｅは、さらに、ＲｅｑＯｂｆとＡｃｃｅｐｔＣｏｓｔとを更新し、終了条件に達しているかどうかを調べる。ｗ１、ｗ２、ｗ３、ｗ４は、インプリメンテーション定義定数である。

$I(S) := I(S) - (w_2 T_{pos}(S) + w_2 T_{res}(S))$;

ReqObf:=ReqObf-($w_2T_{pos}$(S)+$w_2T_{res}$(S));
AcceptCost:=AcceptCost-$T_{cost}$(S);
RETURN AcceptCost<;=0 OR ReqObf<;=0.

アルゴリズム５（プラグマティック情報）

入力

ａ）アプリケーションＡ。

ｂ）Ａに対する入力データセットＩ＝｛Ｉ１；Ｉ２；．．．｝。

出力：

ａ）Ａ内の各ルーチンMに関してMの実行時間ランクを与えるマッピングＲ（M）。

ｂ）Ａ内の各ソースコードオブジェクトＳに関してＳで使用される言語構造体のセットを与える、マッピングＰｓ（Ｓ）。

プラグマティック情報を計算する。この情報が、各々の個別のソースコードオブジェクトに関する情報の適正タイプを選択するために使用されることになる。

１．動的プラグマティック情報を計算する（すなわち、ユーザによって提供される入力データセットＩに基づくプロファイラ下でアプリケーションを実行する）。アプリケーションがその時間の大部分をどこで費やすのかを示す、各ルーチン／基本ブロックに関するＲ（Ｍ）（Mの実行時間ランク）を計算する。

２．静的プラグマティック情報$P_S$（Ｓ）を計算する。$P_S$（Ｓ）が、プログラマがＳで使用した言語構造体の種類に関する統計を

提供する。

ＦＯＲ　Ｓ：＝Ａ　ＤＯにおける各ソースコードオブジェクト

　　Ｏ：＝Ｓが使用する演算子のセット；

　　Ｃ：＝Ｓが使用する高レベル言語構造体（ＷＨＩＬＥ文、例外、スレッド等）のセット；

　　Ｌ：＝Ｓが参照するライブラリクラス／ルーチンのセット；

　　Ｐｓ（Ｓ）－ＯＵＣＵＬ；

ＥＮＤ　ＦＯＲ.

アルゴリズム６（混乱化プライオリティ）
入力：
ａ）アプリケーションＡ。
ｂ）Ｒ（Ｍ）、Ｍのランク。
出力
Ａの各ソースコードオブジェクトＳに関してＳの混乱化プライオリティを与えるマッピングＩ（Ｓ）。
Ｉ（Ｓ）が、ユーザによって明示的に提供されることが可能であるか、または、アルゴリズム５で収集された静的データに基づいた発見法を使用して計算されることが可能である。使用可能な発見法は次の通りであってよい。
１．Ａの任意のルーチンＭに関して、Ｍのランク、すなわち、Ｒ（Ｍ）に対してＩ（Ｍ）を反比例させる。すなわち、この着想は、「ルーチンを実行するのに多くの時間が費やされる場合には、恐らくは、Ｍが、厳重に混乱化されなければならない重要な手続きだろう」ということである。
２．テーブル１のソフトウェア複雑性測度の１つによって定義される通りに、Ｉ（Ｓ）をＳの複雑性とする。この場合も同様に、（

間違っている可能性もある）直感的洞察は、複雑なコードの方が単純なコードの場合よりも重要なトレードシークレットを含む可能性が高いということである。
アルゴリズム７（混乱化の適切性）
入力：
ａ）アプリケーションＡ。
ｂ）変換Ｔの各々に関して、Ｔがアプリケーションに加えるであろう言語構造体のセットを与える、マッピングＰｔ。
ｃ）Ａの各ソースコードオブジエクトＳに関して、Ｓで使用される言語構造体のセットを与えるマッピングＰｓ（Ｓ）。
出力：
Ａの各ソースコードオブジェクトＳと各変換Ｔとに関して、Ｓに対するＴの適切性を与えるマッピングＡ（Ｓ）。

各ソースコードオブジェクトＳに関して適切性セットＡ（Ｓ）を計算する。マッピングは、基本的に、アルゴリズム５で計算された静的プラグマティック情報に基づいている。

```
ＦＯＲ　Ｓ：＝Ａ　ＤＯにおける各ソースコードオブジェクト
    ＦＯＲ　Ｔ：＝各変換ＤＯ
            Ｖ：＝Ｐｔ（Ｔ）とＰｓ（Ｓ）との間の類似性の度合い；
            Ａ（Ｓ）：＝Ａ（Ｓ）Ｕ｛Ｔ－－＞Ｖ｝；
    ＥＮＤ　ＦＯＲ
ＥＮＤ　ＦＯＲ
```

１１．概要と考察

被混乱化プログラムがオリジナルのプログラムとは異った形で動作することが、多くの状況において許容可能であるだろうということを、本発明者は認識している。特に、本発明の混乱化変換の殆どが、そのオリジナルのプログラムに比べて目標プログラムの動作速度を遅くするかまたはプログラムサイズを大きくする。本発明では、特殊な場合に、目標プログラムが、オリジナルのプログラムとは異った副作用を有することさえ可能とされ、または、オリジナルのプログラムがエラー条件で終了する時に目標プログラムが終了しないことさえ可能とされる。本発明の混乱化変換の唯一の必要条件は、これら２つのプログラムの観測可能な動作（ユーザによって経験されるような動作）が同一でなければならないということだけである。

オリジナルのプログラムと被混乱化プログラムとの間のこうした弱い同一性を可能にすることは、新規性のある非常に刺激的な着想である。様々な変換が提供され上述されているが、他の様々な変換が当業者には明らかだろうし、本発明によるソフトウェアセキュリティ増強のための混乱化を実現するために使用されることが可能である。

さらに、未だ知られていない変換を発見するために、様々な将来の調査研究が行われる可能性が大きい。特に、次の領域の調査研究が行われることが期待されている。

１．新たな混乱化変換が発見されるべきである。
２．様々な変換の間での相互関係とオーダリングとが研究されるべきである。これは、最適化変換シーケンスのオーダリングが常に困難な問題である、コード最適化における研究と同様である。
３．効力とコストとの間の関係が研究されるべきである。個々の種類のコードに関して、どの変換が最良の「出費に見合うだけの価値」（すなわち、最低の実行オーバヘッドで最高の効力）を与えるか

を知ることが求められている。

上記の変換の全てを概観するためには、図３１を参照されたい。上記の不明瞭構造体を概観するためには、図３２を参照されたい。しかし、本発明は、上記の典型的な変換と不明瞭構造体とに限定されてはならない。

１１．１　混乱化能力

暗号化とプログラム混乱化は互いに極めて類似している。これらの両方は、秘密を探り出そうとする人間たちの目から情報を隠蔽しようとするだけでなく、限られた時間でこの隠蔽を行うことも意図している。暗号化されたドキュメントは限られた貯蔵寿命を有する。暗号化プログラム自体が攻撃に耐えている限りにおいてだけ、かつ、ハードウェアの処理速度の進歩が、選択されたキー長に関するメッセージが頻繁に解読されることを可能にしない限りにおいてだけ、暗号化されたドキュメントが安全であるにすぎない。被混乱化アプリケーションにも同じことが当てはまる。十分に強力な混乱解除器が未だ構築されていない限りにおいてだけ、被混乱化アプリケーションが安全であるにすぎない。

混乱解除器が混乱化器が追いつくのに要する時間よりもリリースの間の時間が短い限りは、アプリケーションを進化させる上では、このことは問題ではないだろう。混乱解除器が混乱化器に追いついても、アプリケーションが自動的に混乱解除されることが可能となる時点までには、そのアプリケーションが既に時代遅れになっており、したがって競合相手の関心の対象ではなくなっているだろう。

しかし、アプリケーションが、何回かのリリースにわたって存続すると考えられるトレードシークレットを含む場合には、こうしたトレードシークレットが混

乱化以外の手段によって保護されなければならない。部分サーバ側実行（図２（ｂ））がその自明の選択で

あるが、アプリケーションの実行速度が低速となるかまたは（ネットワーク接続がダウンした時には）実行が不可能となるという欠点がある。

１１．２　混乱化の他の使用

上記の通りの混乱化用途以外に他の潜在的な混乱化用途が存在するかも知れないということを指摘することは興味深い。１つの可能性は、ソフトウェアの海賊版製造販売業者を追跡するために混乱化を使用することである。例えば、ベンダが、自分のアプリケーションの新たな被混乱化バージョンを新たな個々の顧客のために作成し（ＳｅｌｅｃｔＴｒａｎｓｆｏｒｍアルゴリズム（アルゴリズム３）の中にランダム性要素を導入することによって、同じアプリケーションの各々に異った被混乱化バージョンが生成されることが可能である。乱数発生器に対する異ったシードが別々のバージョンを生じさせる）、各バージョンを販売した顧客の記録を保持する。そのアプリケーションがネットを通じて販売され配給されている場合にだけ、これはおそらく妥当であるにすぎないだろう。ベンダが自分のアプリケーションの海賊版が販売されていることを発見した場合には、このベンダに必要なことは、その海賊版バージョンのコピーを手に入れて、それをデータベースと比較し、そのオリジナルのアプリケーションを誰が購入したかを発見することだけである。実際は、販売された全ての被混乱化バージョンのコピーを保存することは不要である。販売された乱数シードを保存しておくだけで十分である。

ソフトウェアの海賊版製造販売業者が混乱化を（不正）使用する可能性もある。本明細書で概説されたＪａｖａ混乱化器がバイトコードのレベルで動作するので、合法的に購入されたＪａｖａアプリケーションに対して海賊版製造販売業者が混乱化を施すことを阻止

するものは何もない。その後で、この被混乱化バージョンが再販売されることも可能である。海賊版製造販売業者が訴訟に直面した時には、彼らは、実際には自

分が最初に購入したアプリケーションを再販売しているのではなくて（結局のところ、そのコードが全く異っている！）、合法的に再設計したバージョンを販売しているのであると主張することも可能である。

結論

結論として、本発明は、ソフトウェアのリバースエンジニアリングを防止するかまたは少なくとも妨害するための、コンピュータ上で実現される方法と装置とを提供する。これは、実行時間またはプログラムサイズを犠牲として実現されることが可能であり、その結果として変換されたプログラムが詳細なレベルでは異った形で動作するけれども、本発明の方法が、適切な状況では高い有用性をもたらすものと考えられる。実施様態の１つでは、変換されたプログラムが、非変換のプログラムと見掛け上では同じ動作をする。したがって、本発明は、オリジナルのプログラムと被混乱化プログラムとの間のこうした弱い同一性を可能にする。

本発明の開示が主としてソフトウェアのリバースエンジニアリングの阻止という文脈において行われてきたが、ソフトウェアオブジェクト（アプリケーションを含む）のウォーターマーキング（ｗａｔｅｒｍａｒｋｉｎｇ）ような他のアプリケーションが想定されている。これは、あらゆる単一の混乱化手続きの潜在的に特有の性質を利用する。ベンダは、アプリケーション販売先の個々の顧客に対して、個々に異った被混乱化バージョンを作成するだろう。海賊版コピーが発見された場合には、そのベンダは、その海賊版バージョンをオリジナルの混乱化情報データベースと比較するだけで、オリジナルのアプリケーションを追跡することが可能である。

本明細書で説明されている特定の混乱化変換は、網羅的なものではない。さらに別のタイプの混乱化が、本発明の新規の混乱化ツールアーキテクチャにおいて提供され使用されてよい。

上記の説明では、公知の等効物を有する要素または整数が言及されてきたが、こうした等効物が、上記で個々に説明されたように、本発明に含まれている。

本発明が特定の実施様態を参照して単なる具体例の形で説明されてきたが、本

発明の範囲から逸脱することなしに、改変と改善とが行われることが可能である
ことが理解されなければならない。

【図１】

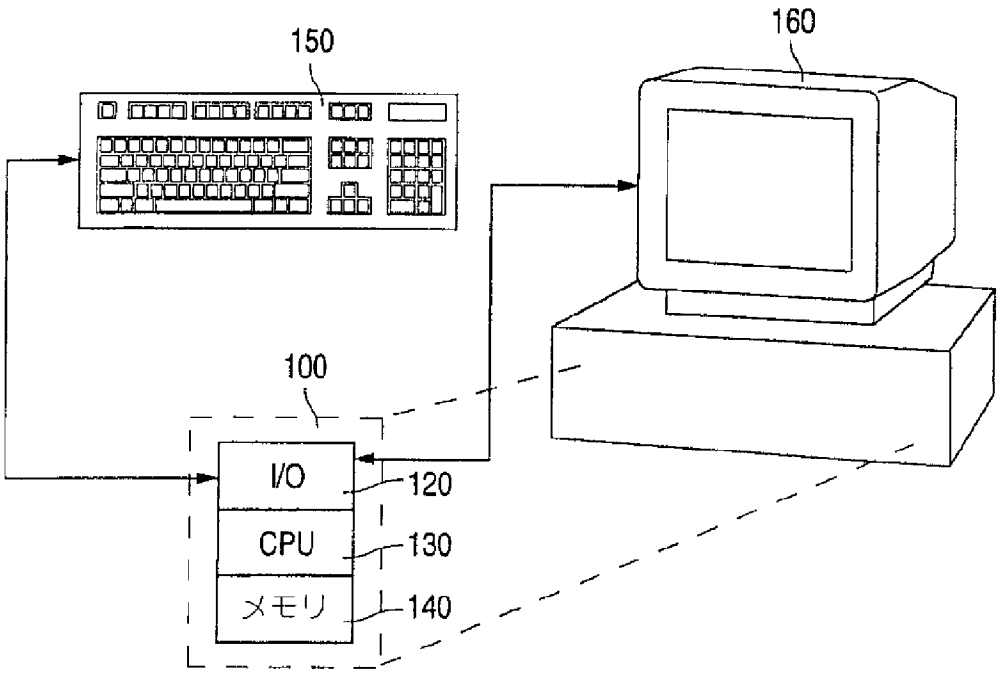



FIG. 1

【図２】

FIG. 2a

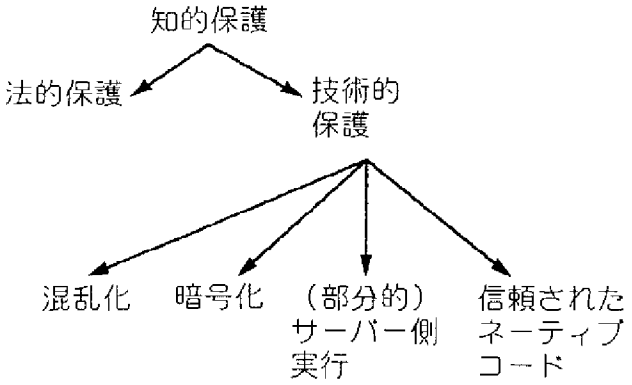
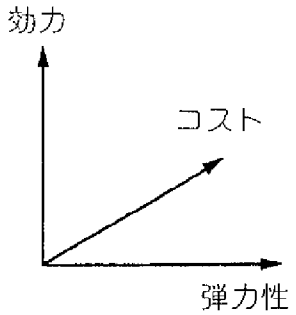



FIG. 2b



【図２】

FIG. 2c

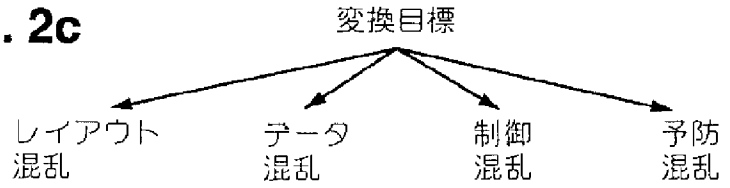



FIG. 2e

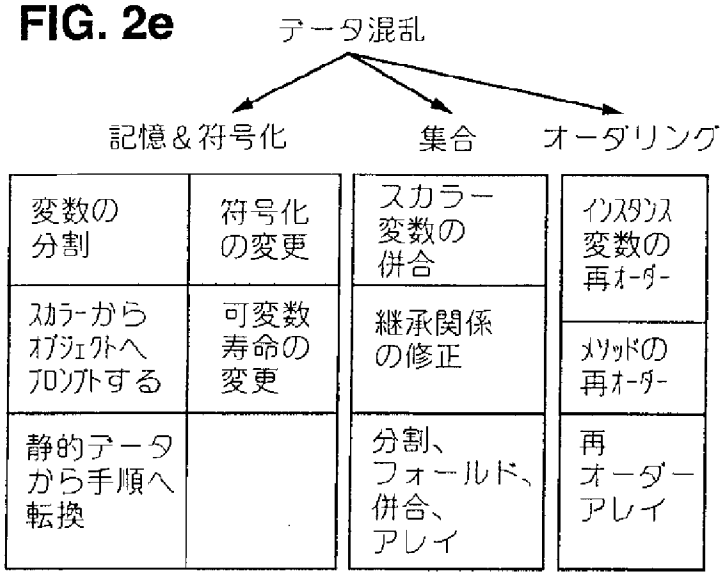
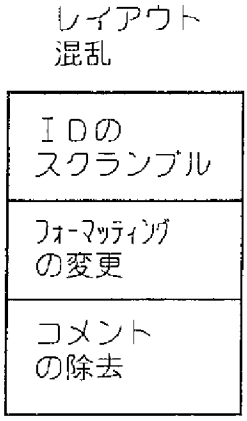



FIG. 2d



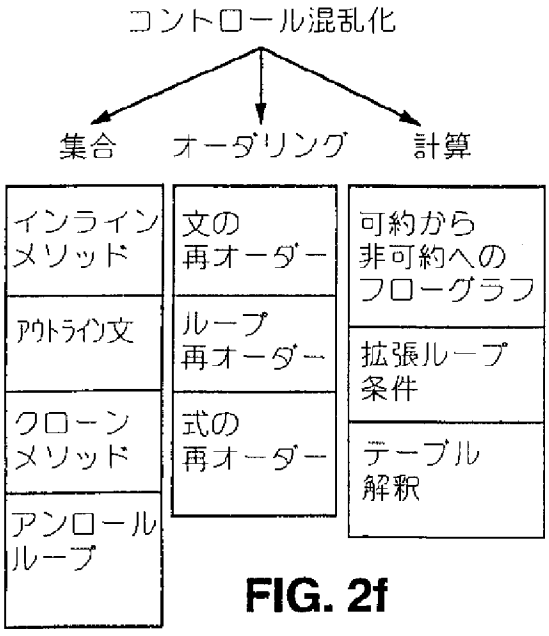




FIG. 2f

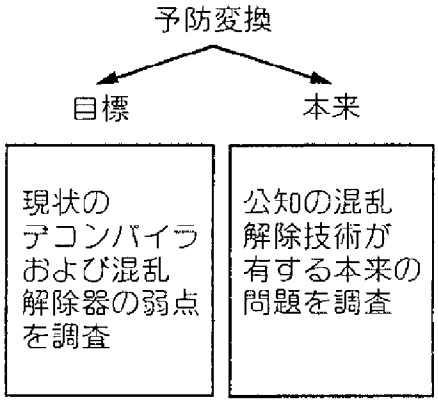



FIG. 2g

【図３】

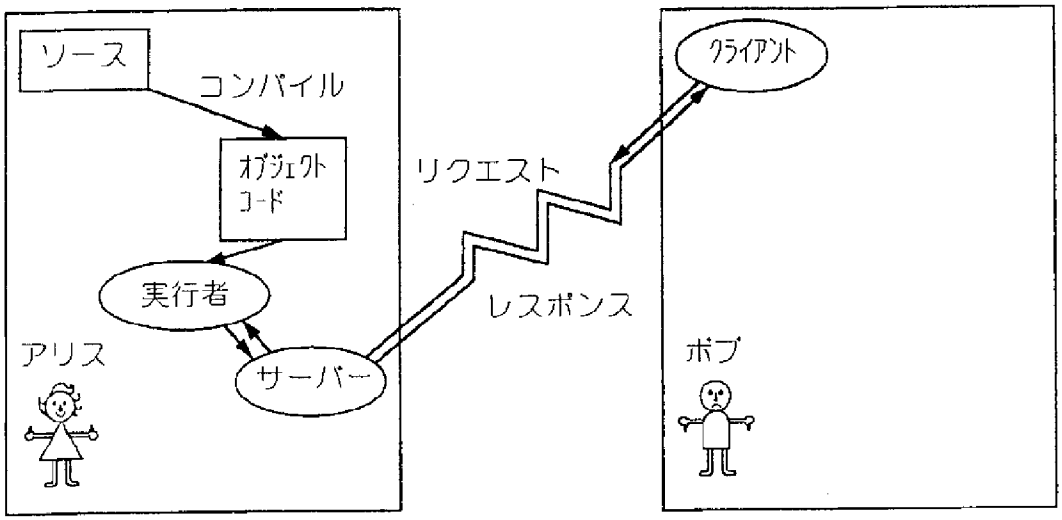



FIG. 3a

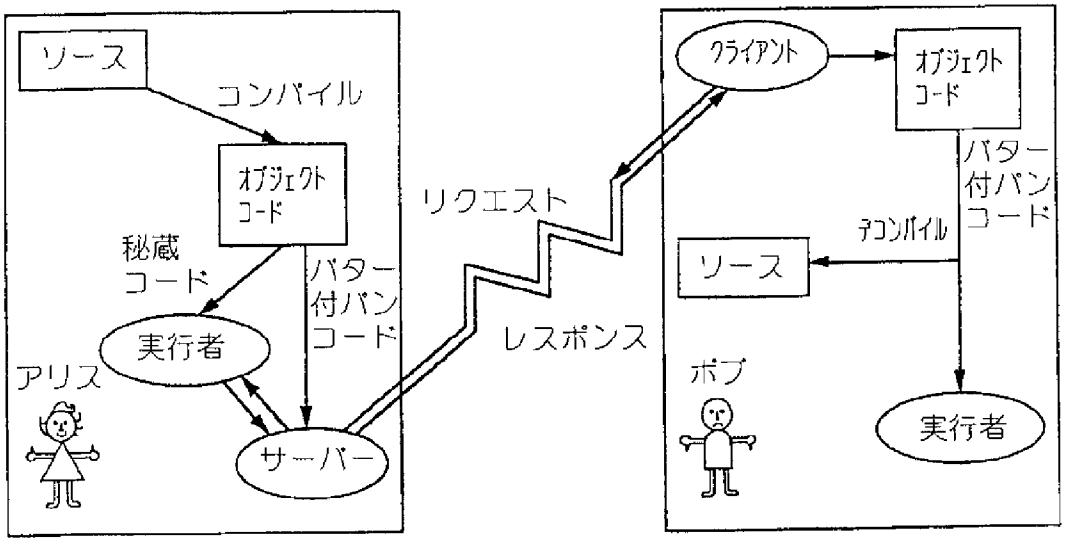



FIG. 3b

【図４】

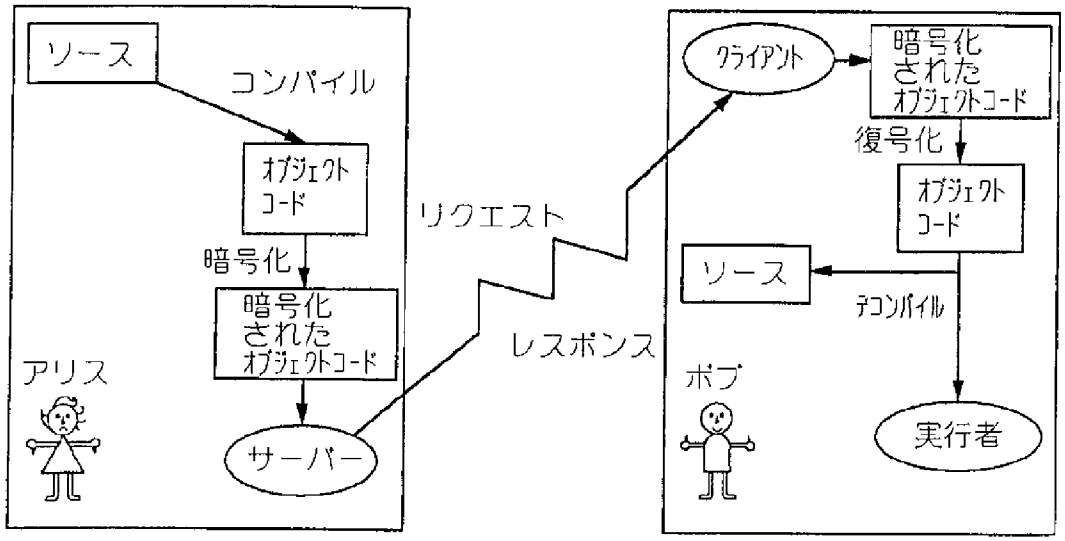



FIG. 4a

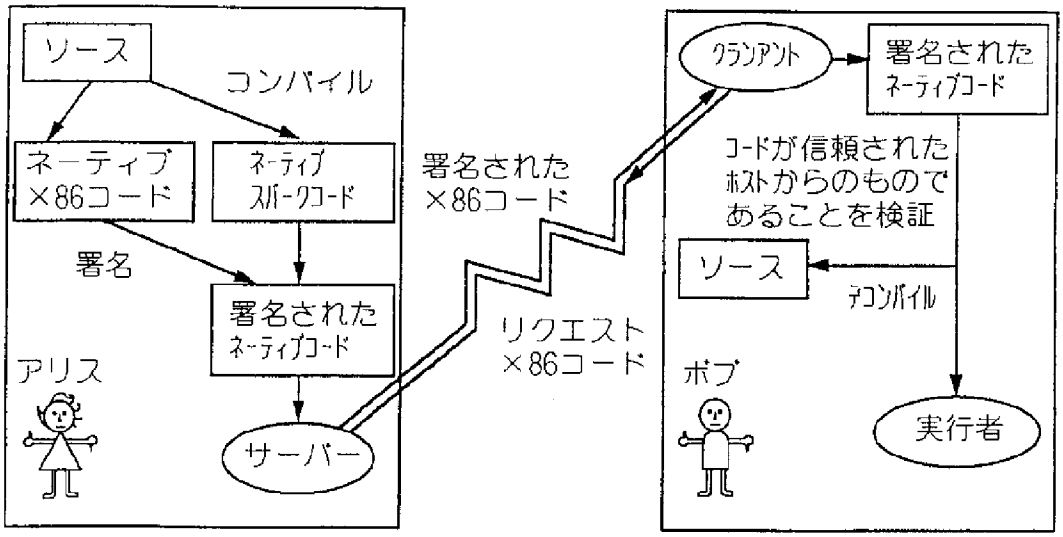



FIG. 4b

【図５】

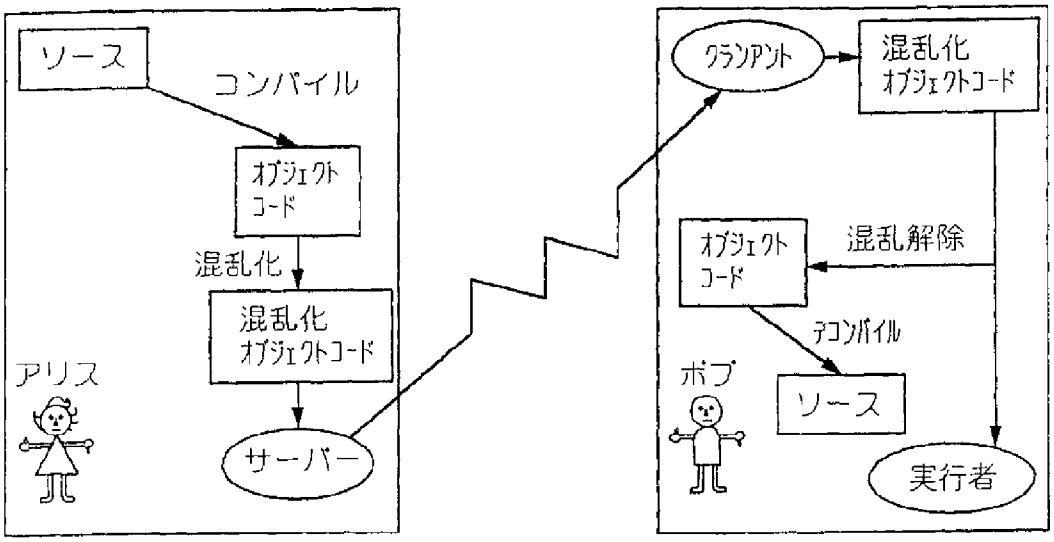

ソース
コンパイル
オブジェクトコード
混乱化
混乱化オブジェクトコード
アリス
サーバー
クランアント
混乱化オブジェクトコード
オブジェクトコード
混乱解除
デコンパイル
ボブ
ソース
実行者

FIG. 5

【図６】

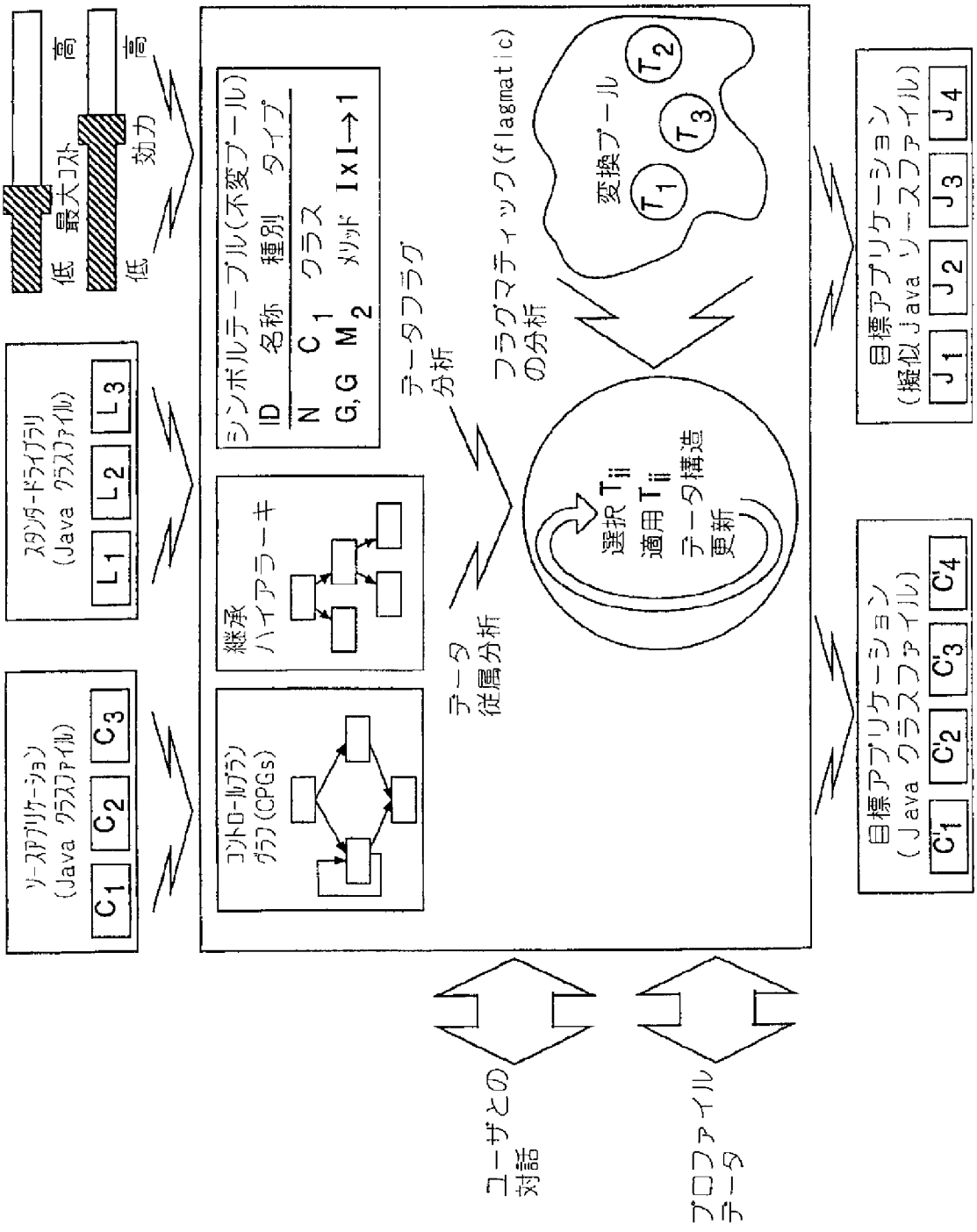

ソースアプリケーション
(Java クラスファイル)
C1 C2 C3
スタンダードライブラリ
(Java クラスファイル)
L1 L2 L3
低 最大コスト 高
低 効力 高
コントロールフローグラフ(CFGs)
継承ハイアラーキ
シンボルテーブル(不変プール)
ID 名称 種別 タイプ
N C1 クラス
G,G M2 メソッド I x I→1
データフラグ
データ分析
データ従属分析
フラグマティック(flagmatic)の分析
変換プール
T1 T2 T3
選択 Tji
適用 Tji
データ構造更新
目標アプリケーション
(Java クラスファイル)
C'1 C'2 C'3 C'4
目標アプリケーション
(擬似 Java ソースファイル)
J1 J2 J3 J4
ユーザとの対話
プロファイルデータ

FIG. 6

【図７】

| メトリック | メトリック名称 | 引用 |
|---|---|---|
| $\mu_1$ | プログラム長 | Halstead |
| | E(P)は、P内のオペランドおよび演算子の数と共に増大する | |
| $\mu_2$ | ｻｲｸﾛﾏﾃｨｯｸ(cyclomatic)の複雑性 | McCabe |
| | E(F)は、F内の述語の数と共に増大する | |
| $\mu_3$ | 入れ子の複雑性 | Harrison |
| | E(F)は、F内条件の入れ子レベルと共に増大する | |
| $\mu_4$ | データフローの複雑性 | Oviedo |
| | E(F)は、F内の内部基本ブロック可変レファレンスの数と共に増大する | |
| $\mu_5$ | ファンーイン／アウトの複雑性 | Henry |
| | E(F)は、Fへの形式パラメータの数と共に、そして、Fにより読まれまたは更新されるグローバルデータ構造の数と共に、増大する | |
| $\mu_6$ | データ構造の複雑性 | Munson |
| | E(P)は、P内にて宣言された静的データ構造の複雑性と共に、増大する。スカラー変数の複雑性は不変である。アレイの複雑性は、ディメンション数と共に、そして、エレメントタイプの複雑性と共に、増大する。レコードの複雑性は、そのフィールドの数および複雑性と共に、増大する。 | |
| $\mu_7$ | OOメトリック | Chidamber |
| | E(C)は、$C_1$内のメソッド数($\mu_7^a$)と、継承木構造内のCの深さ(ルートからの距離)($\mu_7^b$)と、$C_1$のダイレクトサブクラスの数($\mu_7^c$)と、Cが結合される他のクラスの数($\mu_7^d$)と、$C_1$のオブジェクトへ送られたメッセージに応答して実行可能なメソッド数($\mu_7^e$)と、インスタンス変数の同一セットをCのメソッドが参照しない度合($\mu_7^f$)と共に、増大する。注：$\mu_7^f$は結合度、すなわち、モジュールのエレメントがどれ位強く関連しているか、を測る。 | |

*２つのクラスは、もしその一方が他方のメソッドまたはインスタンス変数を使用するならば、結合される。

**FIG. 7**

【図８】

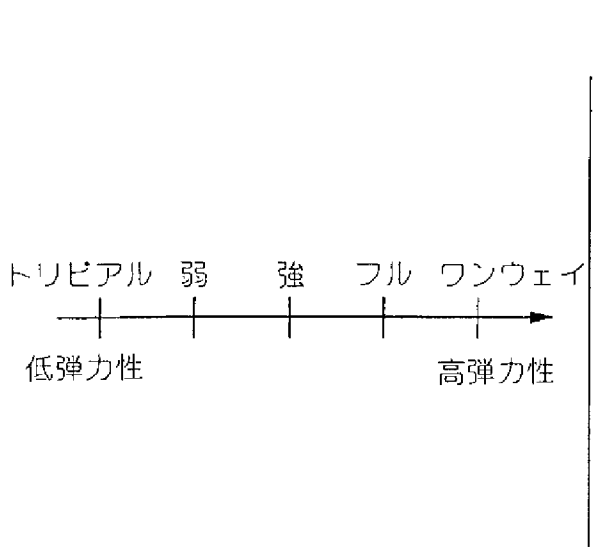



FIG. 8a

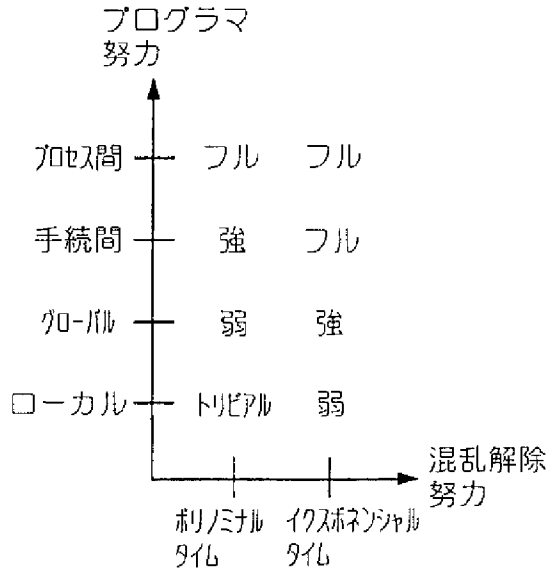



FIG. 8b

【図９】

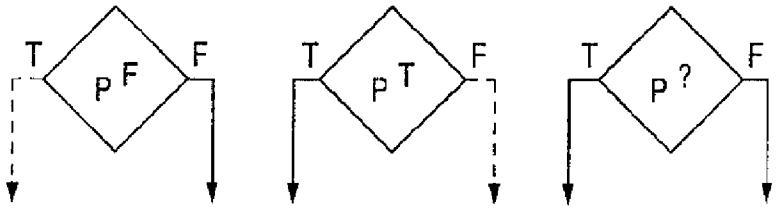



FIG. 9

【図１０】

```
{    int v, a=5; b=6;
     v=11= a + b;
     if (b > 6) T ...
     if (random (1,5) < 0) F...
}
```

FIG. 10a

```
{    int v, a=5; b=6;
     if (...) ...
          ⋮      (b is unchanged)
     if (b < 7) T a++1
     v=11= (a > 5) 7v=b=b; v=b
}
```

FIG. 10b

【図１１】

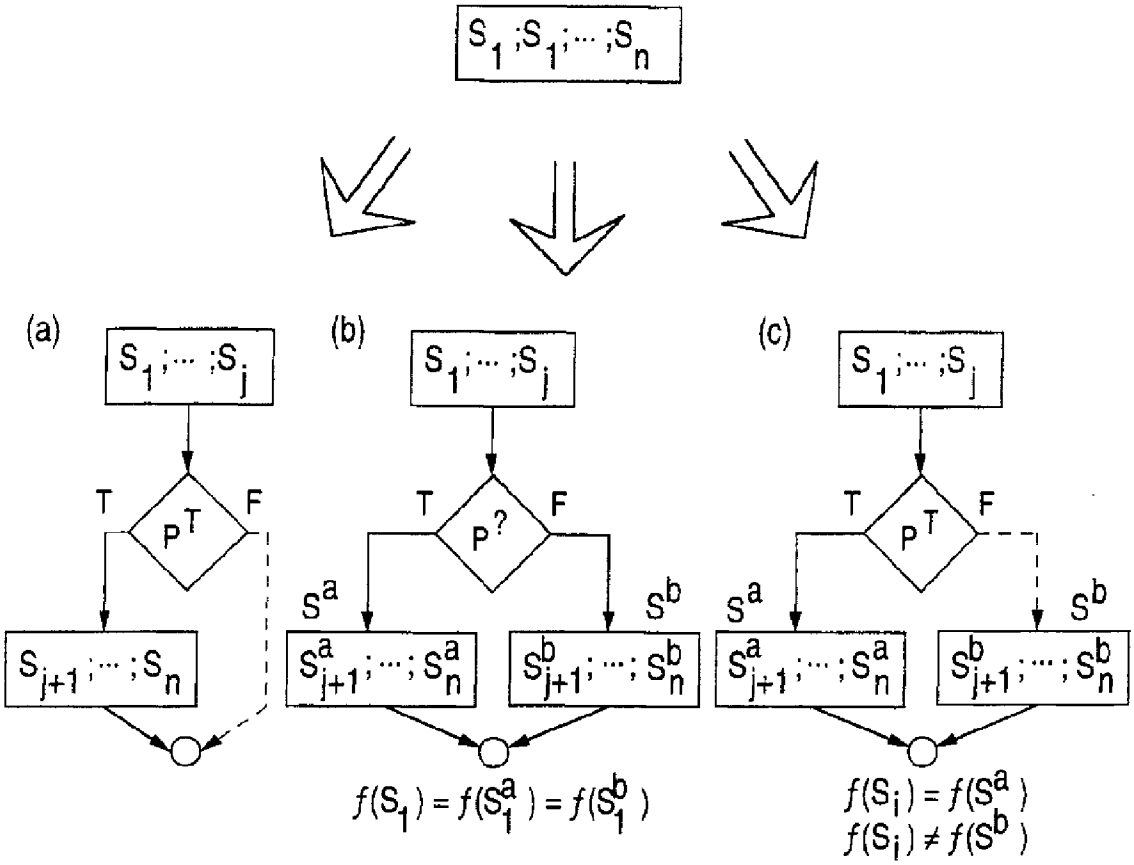
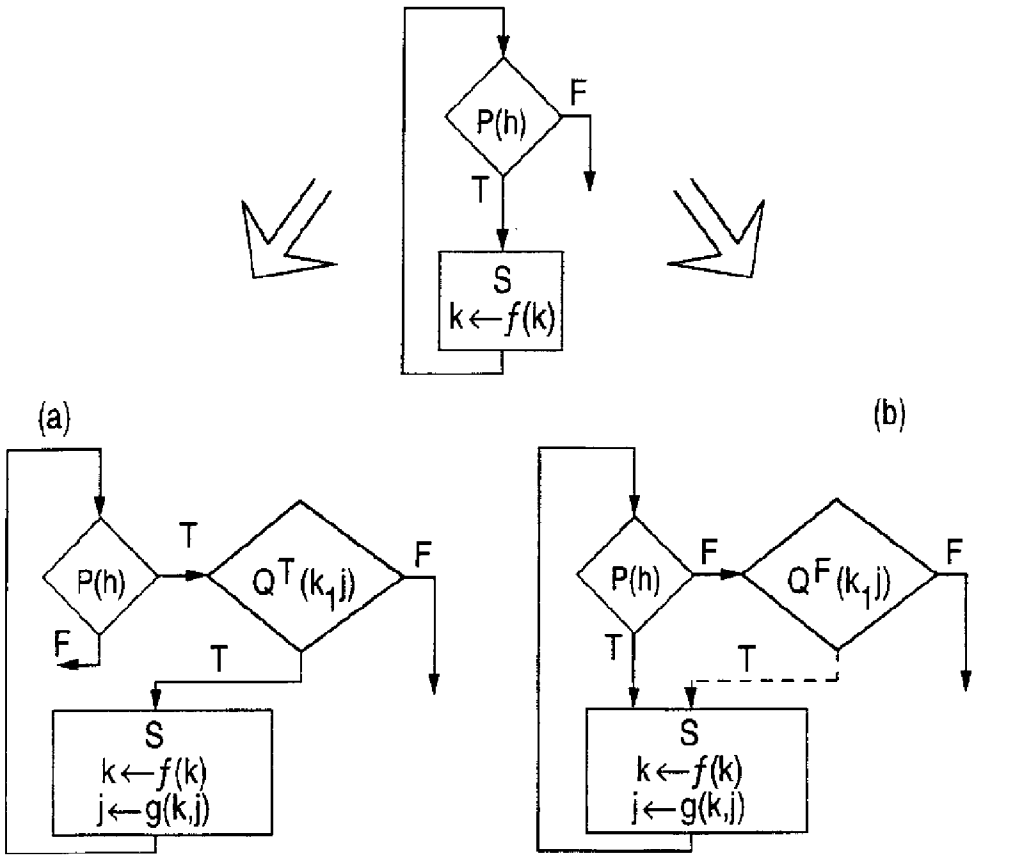

$S_1;S_1;\cdots;S_n$
(a)
$S_1;\cdots;S_j$
T
$P^T$
F
$S_{j+1};\cdots;S_n$
(b)
$S_1;\cdots;S_j$
T
$P^?$
F
$S^a$
$S^b$
$S^a_{j+1};\cdots;S^a_n$
$S^b_{j+1};\cdots;S^b_n$
$f(S_1) = f(S^a_1) = f(S^b_1)$
(c)
$S_1;\cdots;S_j$
T
$P^T$
F
$S^a$
$S^b$
$S^a_{j+1};\cdots;S^a_n$
$S^b_{j+1};\cdots;S^b_n$
$f(S_i) = f(S^a)$
$f(S_i) \neq f(S^b)$

**FIG. 11**

【図１２】




```
i=1
while (1<100) {
        ...
    i++;
}
```

(c)

τ

```
i+1j  j=100;
while ((i<100)ff (j* j * (j+1)* (j + 1)%Δ == 0)^T ) {
        ...
    i++;
    j = j * 1 + 3;
}
```

(d)

**FIG. 12**

【図１３】

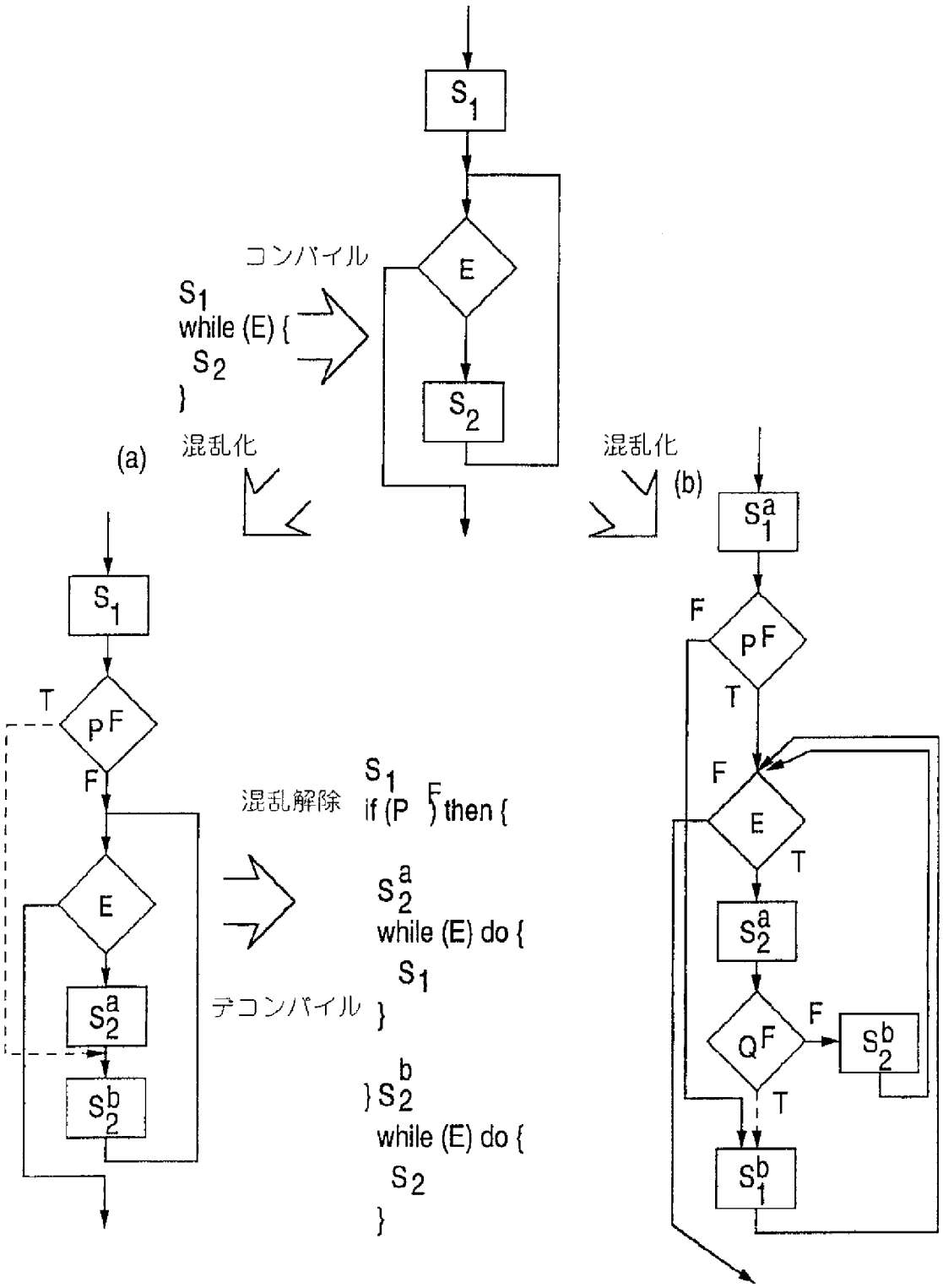

S1
E
S2
コンパイル
S1
while (E) {
S2
}
(a)
混乱化
(b)
混乱化
T
P F
F
E
混乱解除
デコンパイル
if (P F) then {
while (E) do {
while (E) do {
F
T
Q F

FIG. 13

【図１４】

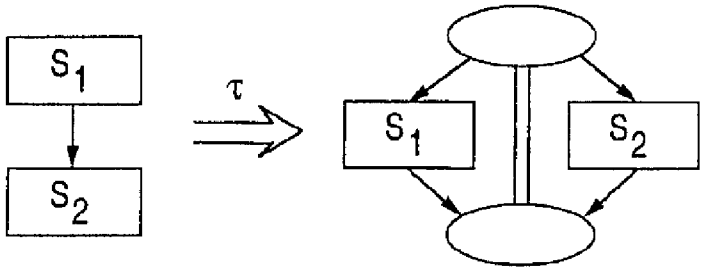



FIG. 14

【図１５】

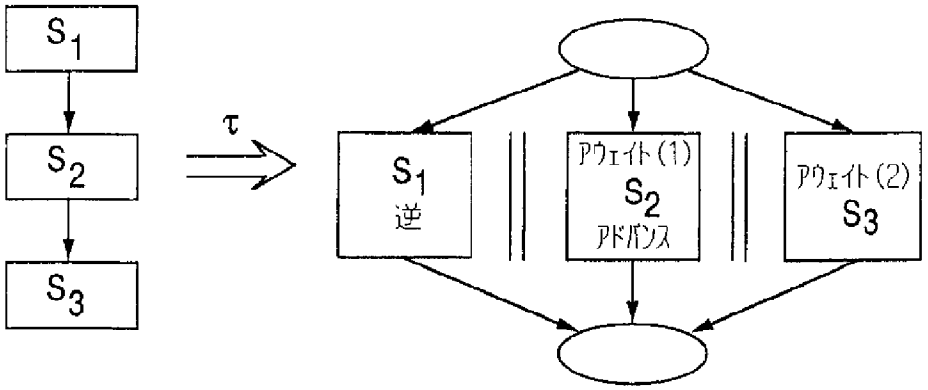



FIG. 15

【図１６】

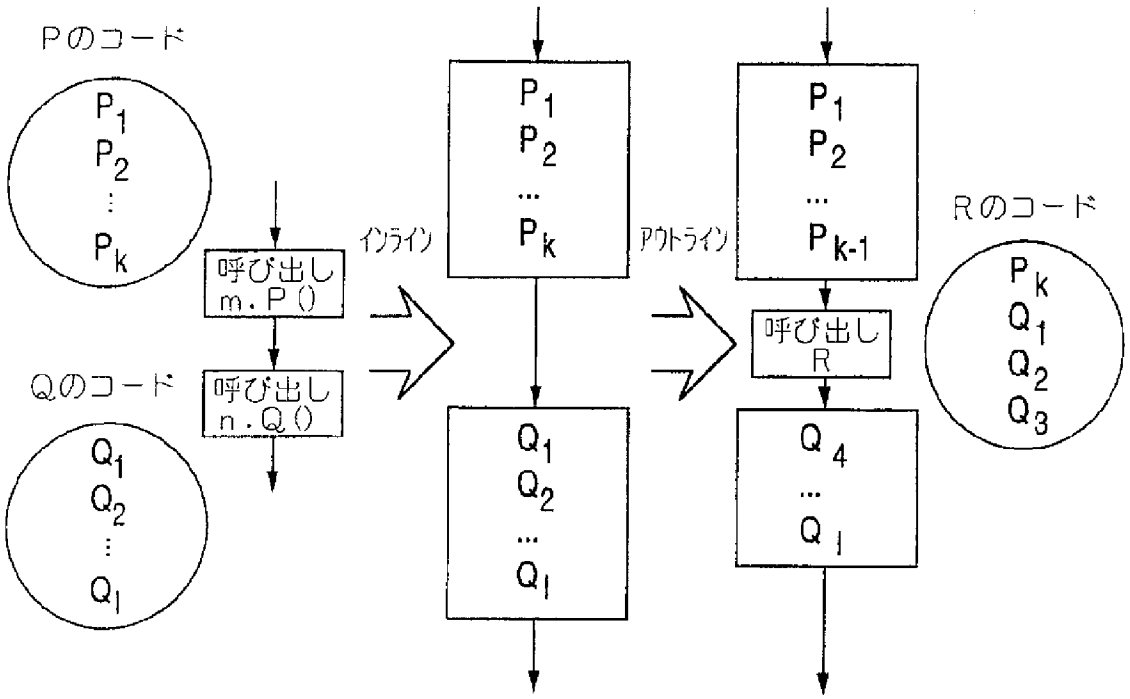



FIG. 16

【図１７】

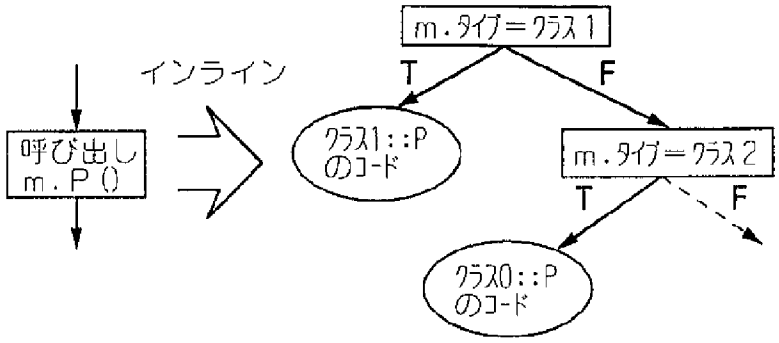



FIG. 17

【図１８】

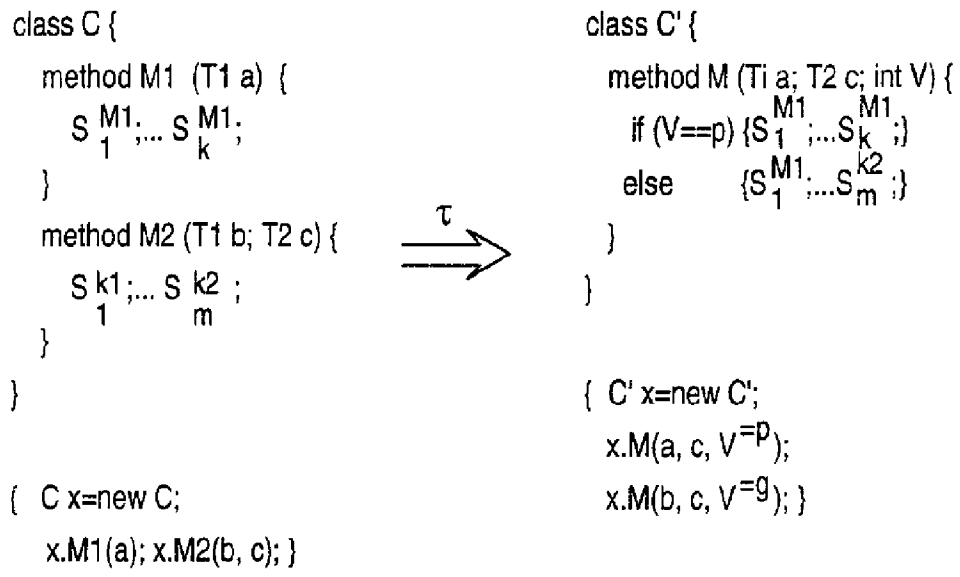



**FIG. 18**

【図１９】

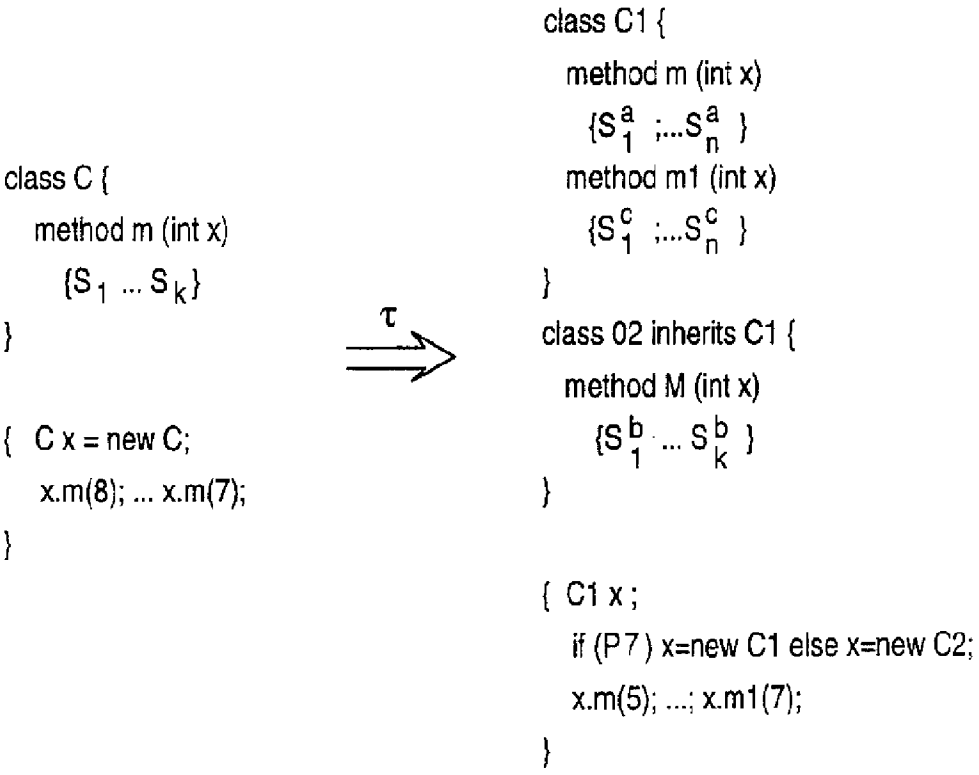



**FIG. 19**

【図２０】

**FIG. 20a**

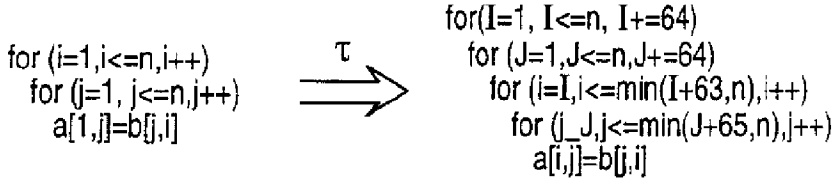



**FIG. 20b**

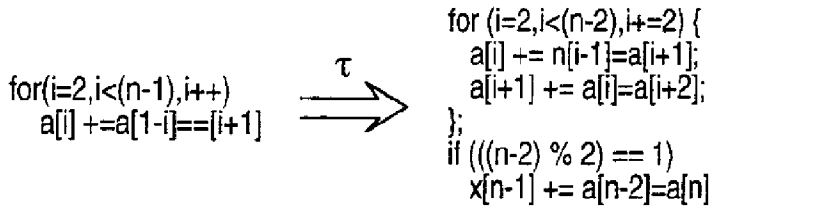
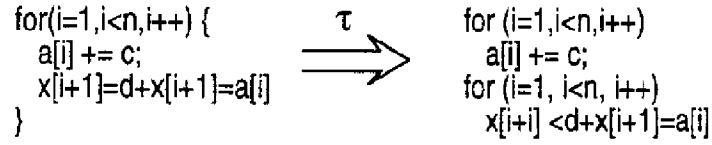



**FIG. 20c**



【図２１】

**FIG. 21a**

| g(V) p | g(V) q | f(p,q) V | 2p + q |
|---|---|---|---|
| 0 | 0 | False | 0 |
| 0 | 1 | True | 1 |
| 1 | 0 | True | 2 |
| 1 | 1 | False | 3 |

**FIG. 21b**

| VAL[p,q] | | p = 0 | p = 1 |
|---|---|---|---|
| q | 0 | 0 | 1 |
| | 1 | 1 | 0 |

**FIG. 21c**

| AND[A,B] | | A = 0 | A = 1 | A = 2 | A = 3 |
|---|---|---|---|---|---|
| B | 0 | 3 | 0 | 0 | 0 |
| | 1 | 3 | 1 | 2 | 3 |
| | 2 | 0 | 2 | 1 | 3 |
| | 3 | 3 | 0 | 0 | 3 |

**FIG. 21d**

| OR[A,B] | | A = 0 | A = 1 | A = 2 | A = 3 |
|---|---|---|---|---|---|
| B | 0 | 3 | 1 | 2 | 3 |
| | 1 | 1 | 1 | 2 | 2 |
| | 2 | 2 | 2 | 1 | 1 |
| | 3 | 0 | 1 | 2 | 0 |

```
(1)   bool A,B,C;                    (1')  short a1,a2,b1,b2,c1,c2;
(2)   A = True;                      (2')  a1=0; a2=1;
(3)   B = False;                     (3')  b1=0; b2=0;
(4)   C = False;                     (4')  c1=1; c2=1;
(5)   C = A & B;          T          (5')  x=AND[2*a1+a2,2*b1+b2]; c1=x/2; c2=x%2;
(6)   C = A & B;                     (6')  c1=(a1 ^ a2) & (b1 ^ b2); c2=0
(7)   C = A | B;                     (7')  x=OR[2*a1+a2,2*b1+b2]; c1=x/2; c2=x%2;
(8)    if (A) ...;                   (8')  x=2*a1+a2; if ((x==1) || (x==2) ...;
(9)    if (B) ...;                   (9')  if (b1 ^ b2) ...;
(10)  if (C) ...;                    (10') if (VAL[c1,c2]) ...;
```

**FIG. 21e**

【図２２】

```
String G (int n) {
    int i=0,k;
    String B;
    while (i) {
        L1:  if (n==1) {S[i++]="A";k=0;goto L6};
        L2:  if (n==2) {S[i++]="B";k= -2 ;goto L6};
        L3:  if (n==3) {S[i++]="C";goto L8};
        L4:  if (n==4) {S[i++]="K";goto L9};
        L5:  if (n==5) {S[i++]="C";goto L11};
               if (n>12) goto L1;
        L6:  if (k++<=2) {S[i++]="A";goto L6} else goto L8;
        L8:  return S;
        L9:  S[i++]="C"; goto L10;
        L10:  S[i++]="B"; goto L8:
        L11:  S[i++]="C"; goto L12;
        L12:  goto L10;
    }
}
```

FIG. 22

【図２３】

FIG. 23a

$$Z(X + r,Y) = 2^{32} \cdot Y + (r + X) = Z(X,Y) + r$$
$$Z(X,Y + r) = 2^{32} \cdot (Y + r) + X = Z(X,Y) + r \cdot 2^{32}$$
$$Z(X \cdot r,Y) = 2^{32} \cdot Y + X + r = Z(X,Y) + (r - 1) \cdot X$$
$$Z(X,Y \cdot r) = 2^{32} \cdot Y \cdot r + X = Z(X,Y) + (r - 1) \cdot 2^{32} \cdot Y$$

FIG. 23b

```
(1)  int X=45, Y=95;          (1')  long Z=167759066119551045;
(2)  X += 5;          τ       (2')  Z += 5;
(3)  Y += 11;         ⟹      (3')  Z += 47244640256;
(4)  X *= c;                  (4')  z += (c-1)* (Z & 4294967295);
(5)  Y *= d;                  (5')  Z += (d-1)* (Z & 18446744069414584320);
```

【図２４】

```
(1)  int A[9];
(2)  A[i] = ...;
       ...
       ...
(3)  int B[8],C[19];
(4)  B[i] = ...;
(5)  C[i] = ...;
       ...
(6)  int D[9]
(7)  for(i=o;i<=B;i++)
       D[i]=2 * D[i+1];
       ...
       ...
       ...
(8)  int E[2,2];
(9)  for(i=Q;i<=2;i ++)
       for(j=0;i<=2;i++)
         swap(E(i,j], E[j,i]);
```

τ ⟹

```
(1')  int A1[4],A2[4];
(2')  if ((i%2)==0) A1[i/2] =...
      else A2[i/2]=...;
        ...
(3')  int BC[20];
(4')  BC[3*i] = ...;
(5')  BC[i/2*3+1+i%2] = ...;
(6')  int D1[1,4];
(7')  for(j=0;j<=1;j++)
        for(k=0;k<=4;k++)
          if (k==4)
            D1[j,k]=2*D1[j+1,0];
          else
            D1[j,k]=2*D1[j,k+1];
        ...
(8')  int E1[8]
(9')  for(i=0;1<=8;i++)
        swap(E[i], E[3=(i%3)+i/3]);
```

**FIG. 24a**

【図２４】

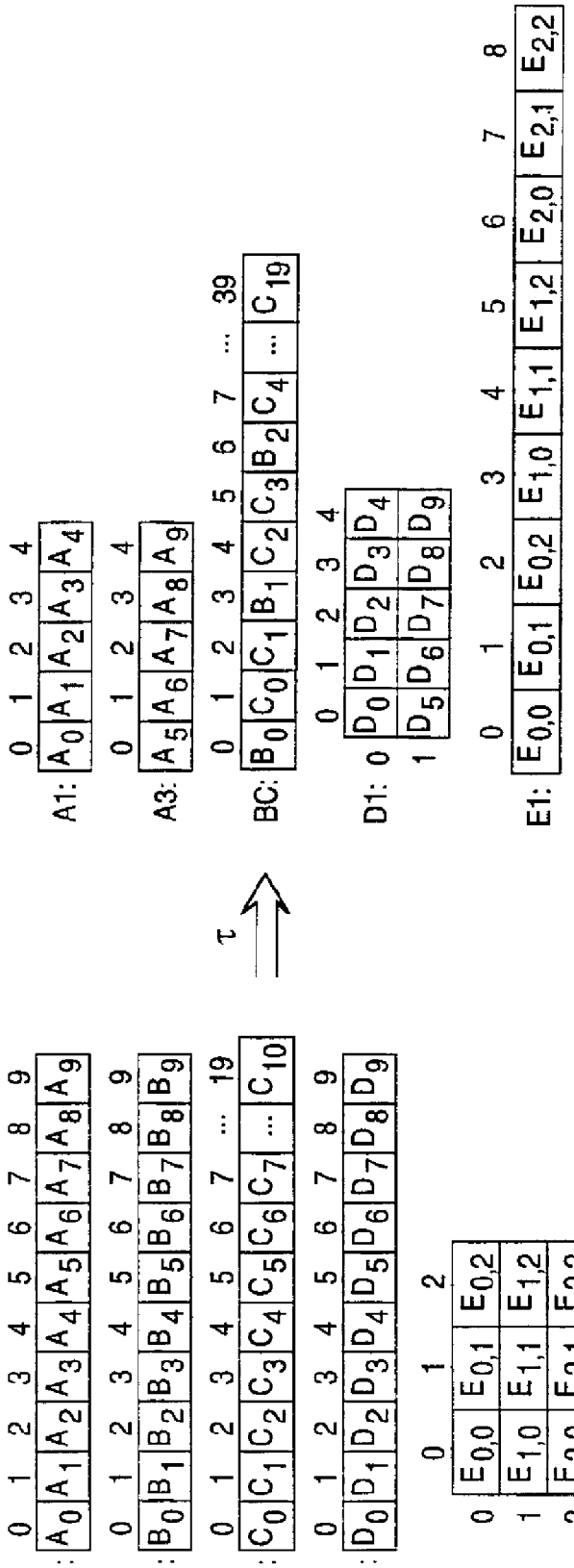



FIG. 24b

【図２５】

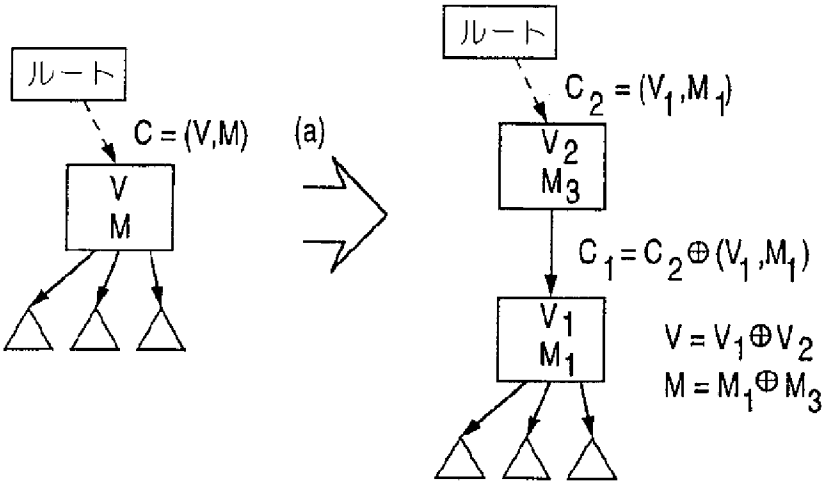



FIG. 25a

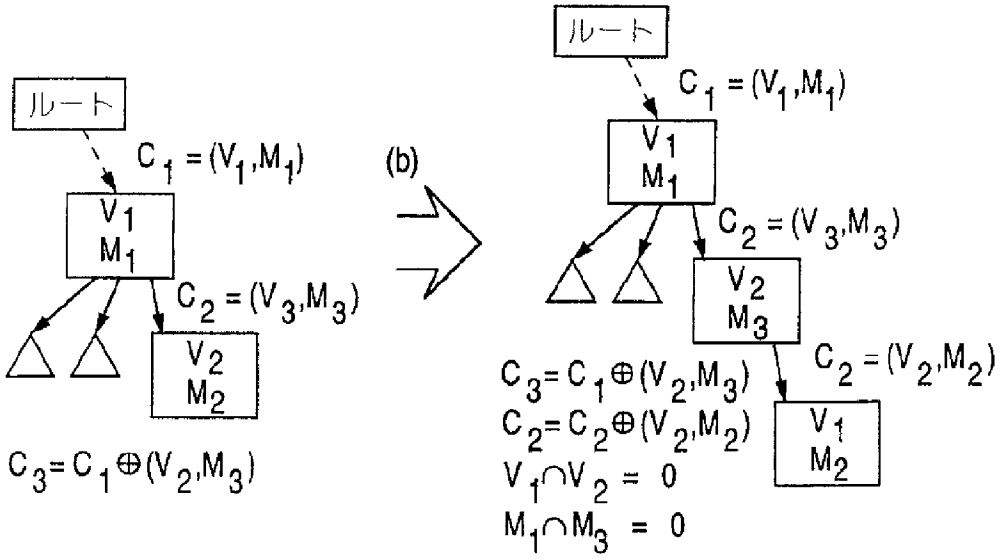



FIG. 25b

【図２５】

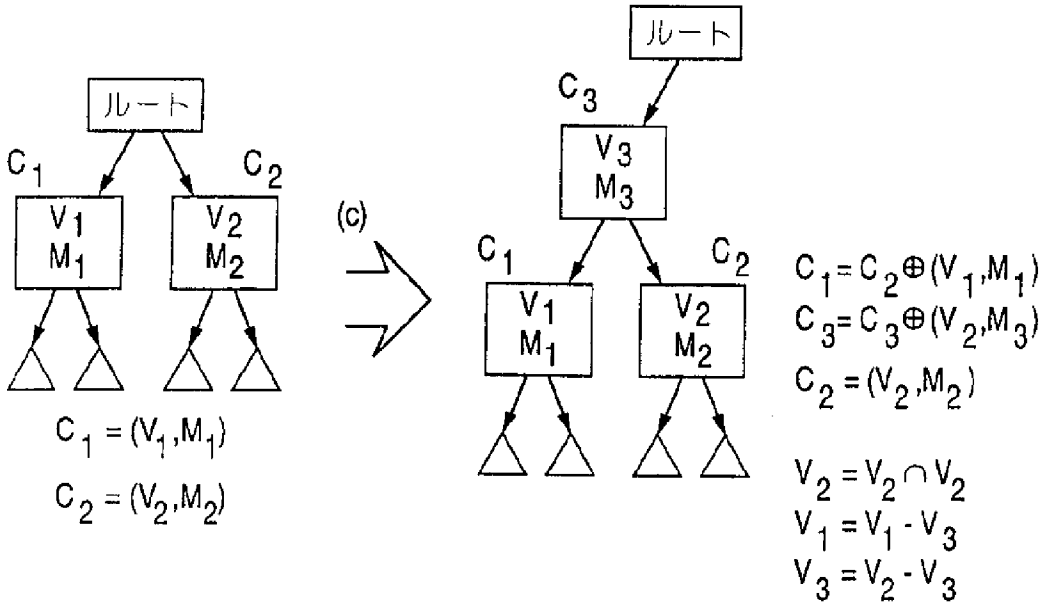



FIG. 25c

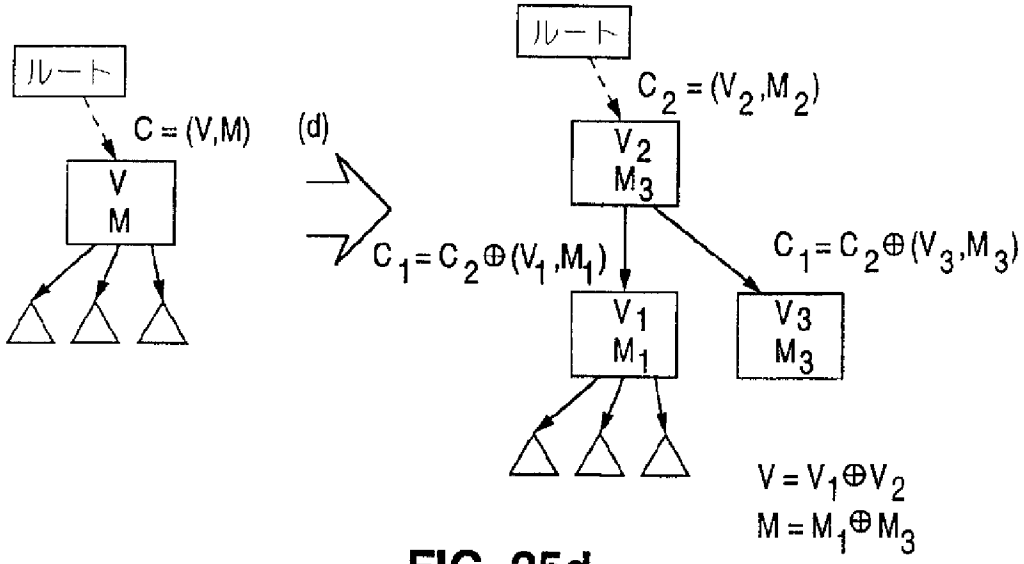
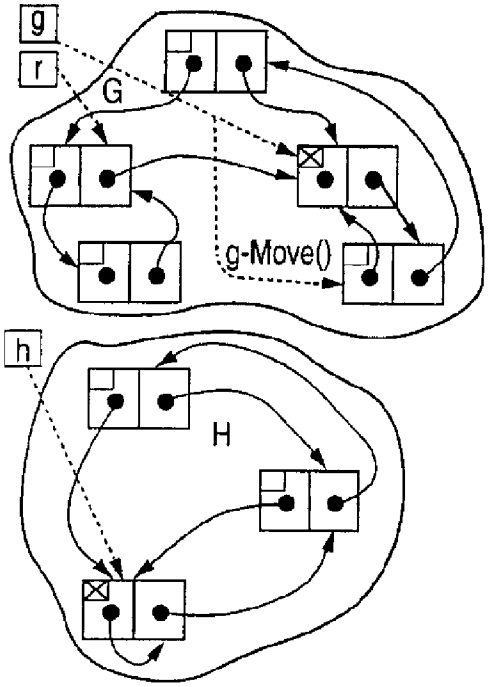



FIG. 25d

【図２６】




```
Node g, h;
method P(...,Node f) {
  /* 1 */ g = g.Move();
          h = h.Move();
  /* 2 */ h = h.Insert(new Node);
          ⋮
  /* 3 */ x.R(...,f.Move());
          ⋮
  /* 4 */ if (f==g) ?...

  /* 5 */ if (g==h) F...
          ⋮
  /* 6 */ f.Token=False;
          g.Token=True;

  /* 7 */ if (f.Token)? ...
          ⋮
  /* 8 */ f.Token=True;
          h.Token=False;

  /* 9 */ if (f.Token) T ...
}
```

FIG. 26

【図３０】

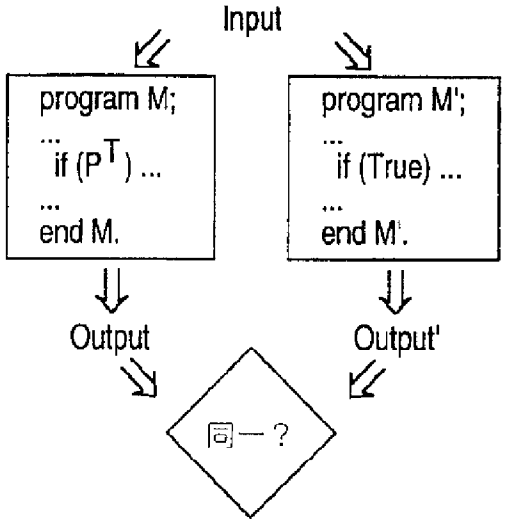



FIG. 30

【図２７】

```
thread S {
  int R;
  while (1) {
    R = random(1,C);
    X = R*R;
    sleep(3);
  }
}
```

```
thread T {
  int R;
  while (i) {
    R = random(1,C);
    X = 7*R*R;
    sleep(2);
    X *= X;
    sleep(5);
  }
}
```

```
int X, Y;
const C = sqrt(maxint)/10;
main () {
  S.run(); T.run();
      ...
  if ((Y - 1) == X) F <= P
      ...
}
```

**FIG. 27**

【図２８】

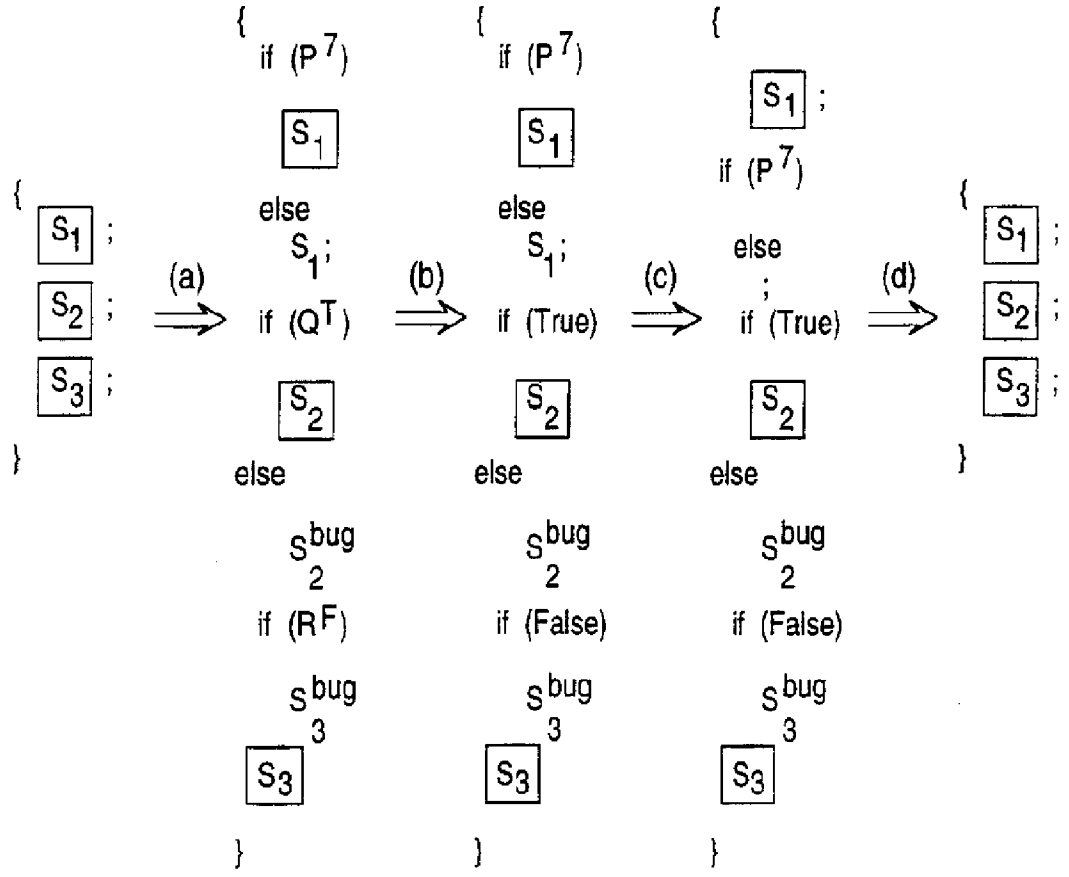



**FIG. 28**

【図２９】

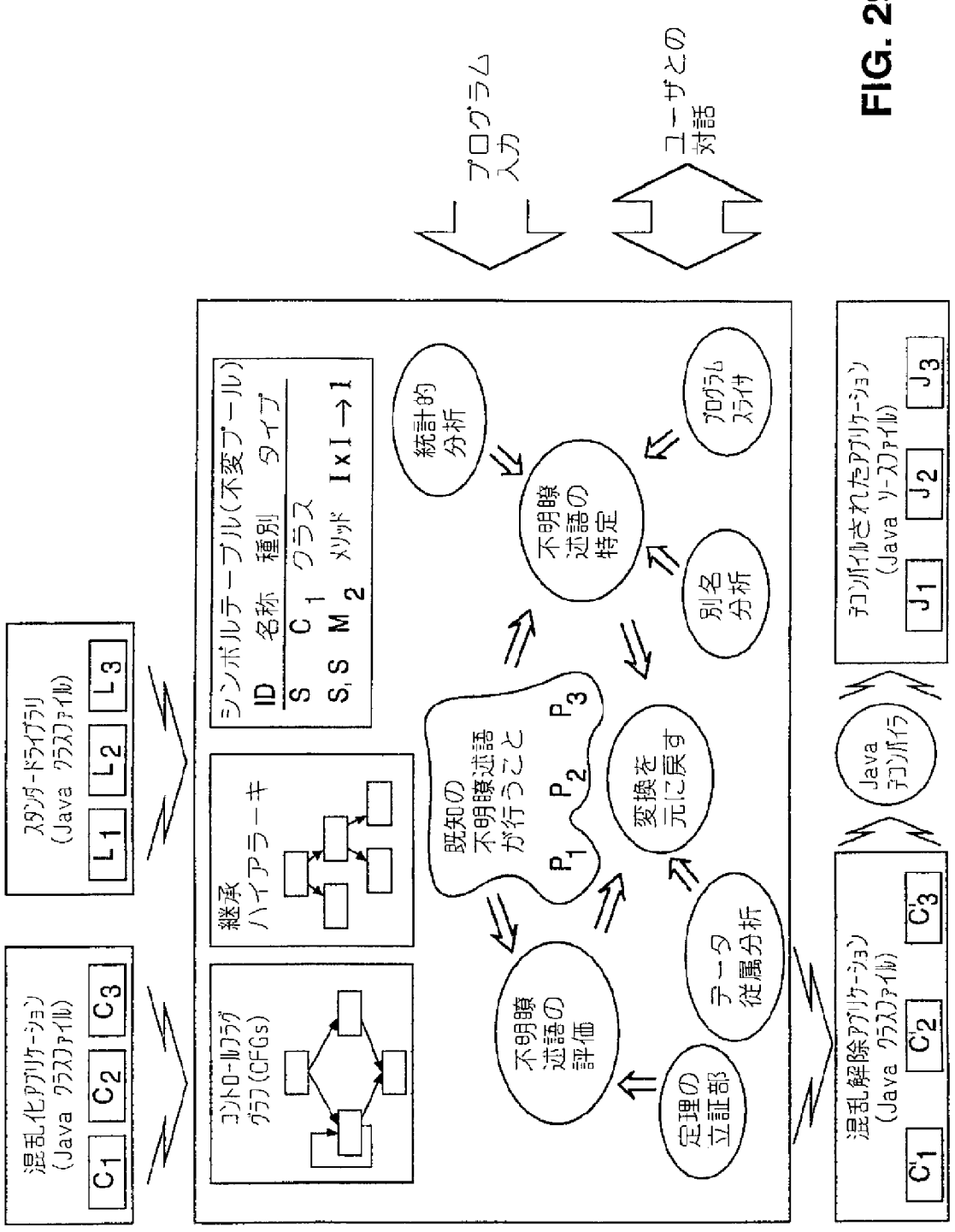

混乱化アプリケーション
(Java クラスファイル)
C1 C2 C3
スタンダードライブラリ
(Java クラスファイル)
L1 L2 L3
コントロールフラグ
グラフ(CFGs)
継承
ハイアラーキ
シンボルテーブル(不変プール)
ID 名称 種別 タイプ
S C1 クラス
S,S M2 メソッド I x I → I
既知の
不明瞭述語
が行うこと
P1 P2 P3
不明瞭
述語の
評価
定理の
立証部
データ
従属分析
変換を
元に戻す
統計的
分析
不明瞭
述語の
特定
別名
分析
プログラム
スライサ
プログラム
入力
ユーザとの
対話
混乱解除アプリケーション
(Java クラスファイル)
C'1 C'2 C'3
Java
デコンパイラ
デコンパイルされたアプリケーション
(Java ソースファイル)
J1 J2 J3

FIG. 29

【図３１】

| 混乱化 目標 | 操作 | 変換 | 品質 効力 | 弾力性 | コスト | メトリクス | セクション |
|---|---|---|---|---|---|---|---|
| レイアウト | | ＩＤのスクランブル | 中 | ワンウェイ | 無料 | | 5.5 |
| | | ﾌｫｰﾏｯﾃｨﾝｸﾞの変更 | 低 | ワンウェイ | 無料 | | 5.5 |
| | | コメントの除去 | 高 | ワンウェイ | 無料 | | 5.5 |
| コントロール | 計算 | デッド又はイレリバントコードの挿入 | 不明瞭述語の品質と、コンストラクトが挿入される入れ子深さとに依存する | | | μ1, μ2, μ3 | 6.2.1 |
| | | ループ条件の拡張 | | | | μ1, μ2, μ3 | 6.2.2 |
| | | 可約から非可約へ | | | | μ1, μ2, μ3 | 6.2.3 |
| | | 冗長オペランドの付加 | | | | μ1 | 6.2.6 |
| | | プログラム用イディオムの除去 | 中 | 強 | † | μ1 | 6.2.4 |
| | | テーブル解釈 | 高 | 強 | 高価 | μ1 | 6.2.5 |
| | 集合 | インラインメソッド | 中 | ワンウェイ | 無料 | μ1 | 6.3.1 |
| | | アウトライン文 | 中 | 強 | 無料 | μ1 | 6.3.1 |
| | | インタリーブメソッド | 不明瞭述語の品質に依存する | | | μ1, μ2, μ3 | 6.3.2 |
| | | クローンメソッド | | | | μ1, μ7 | 6.3.3 |
| | | ブロックループ | 低 | 弱 | 無料 | μ1, μ2 | 6.3.4 |
| | | アンロールループ | 低 | 弱 | 安価 | μ1 | 6.3.4 |
| | | ループ分裂 | 低 | 弱 | 無料 | μ1, μ2 | 6.3.4 |
| | オーダリング | 再オーダー文 | 低 | ワンウェイ | 無料 | | 6.4 |
| | | 再オーダーループ | 低 | ワンウェイ | 無料 | | 6.4 |
| | | 再オーダー式 | 低 | ワンウェイ | 無料 | | 6.4 |

**FIG. 31a-1**

【図３１】

スカラーからオブジェクトへプロンプトする

| 目標 | 操作 | 混乱化 変換 | 品質 効力 | 弾力性 | コスト | メトリクス | セクション |
|---|---|---|---|---|---|---|---|
| データ | 記憶 & 符号化 | 符号化の変更 | 符号化関数の複雑性に依存する | | | μ1 | 7.1.1 |
| | | | 低 | 強 | 無料 | | 7.1.2 |
| | | 可変寿命の変更 | 低 | 強 | 無料 | μ4 | 7.1.2 |
| | | 変数の分割 | 原変数が分割される変数の数に依存する | | | μ1 | 7.1.3 |
| | | 静的データから手続データへの転換 | 生成関数の複雑性に依存する | | | μ1,μ2 | 7.1.4 |
| | 集合 | 併合スカラー変数 | 低 | 弱 | 無料 | μ1 | 7.2.1 |
| | | ファクタクラス | 中 | † | 無料 | μ1, μ7b,c,e | 7.2.3 |
| | | 挿入Bogus クラス | 中 | † | 無料 | μ1, μ7b,c | 7.2.3 |
| | | 再ファクタクラス | 中 | † | 無料 | μ1, μ7b,c,e | 7.2.3 |
| | | 分割アレイ | † | 弱 | 無料 | μ1, μ2, μ6 | 7.2.2 |
| | | 併合アレイ | † | 弱 | 無料 | μ1, μ3 | 7.2.2 |
| | | フォールドアレイ | † | 弱 | 安価 | μ1,μ2,μ3,μ6 | 7.2.2 |
| | | 平担化アレイ | † | 弱 | 無料 | | 7.2.2 |
| | オーダリング | 再オーダーメソッド& インスタンス変数 | 低 | ワンウェイ | 無料 | | 7.3 |
| | | 再オーダーアレイ | 低 | 弱 | 無料 | | 7.3 |

**FIG. 31a-2**

【図３１】

| 目標 | 操作 | 混乱化 変換 | 効力 | 品質 弾力性 | コスト | メトリクス | セクション |
|---|---|---|---|---|---|---|---|
| 目標となる | 予防 | HoseMocha | 低 | トリビアル | 無料 | μ1 | 9 |
| | | スライシングを予防するために別名化形式を付加する | 中 | 強 | 無料 | μ1,μ5 | 9.4 |
| | 本来 | スライシングを予防するために変数従属性を付加する | 不明瞭述語の品質に依存する | | | μ1 | 9.4 |
| | | Bogus データ従属性を付加する | 中 | 弱 | 安価 | μ1 | 9.1.1 |
| | | サイド効果を有する不明瞭な述語を使用する | 中 | 弱 | 無料 | μ1 | 9.5 |
| | | ディク加定理を用いて不明瞭な述語を作る | † | † | † | μ1 | 9.5 |

FIG. 31b

【図３２】

| 不明瞭構造 | 品質 | | セクション |
|---|---|---|---|
| | 弾力性 | コスト | |
| ライブラリ関数の呼び出しから生成される | トリビアル | ライブラリ関数のコストに依存する | 6.1.1 |
| ローカル（内部基本ブロック）情報から生成される | トリビアル | 無料...安価 | 6.1.1 |
| グローバル（内部基本ブロック）情報から生成される | 弱 | 無料...安価 | 6.1.1 |
| 内部手続並びに別名情報から生成される | フル | 安価...高価 | 8.1 |
| プロセス相互作用並びにスケジューリングから生成される | フル | 安価...高価 | 8.2 |

FIG. 32

【国際調査報告】

# INTERNATIONAL SEARCH REPORT

Inter- ional Application No
PCT/US 98/12017

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category ° | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A | WO 97 04394 A (DRAKE CHRISTOPHER NATHAN)<br>6 February 1997<br>see page 3, line 25 - page 4, line 10<br>see page 5, line 25 - page 6, line 8 | 1-20 |
| A | COHEN F B: "OPERATING SYSTEM PROTECTION THROUGH PROGRAM EVOLUTION"<br>COMPUTERS & SECURITY INTERNATIONAL JOURNAL DEVOTED TO THE STUDY OF TECHNICAL AND FINANCIAL ASPECTS OF COMPUTER SECURITY,<br>vol. 12, no. 6, 1 October 1993, pages 565-584, XP000415701<br>see the whole document | 1-20 |
| P,A | WO 97 33216 A (NORTHERN TELECOM LTD)<br>12 September 1997<br>see page 8, line 13 - page 9, line 33 | 1-20 |

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publicationdate of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

| Date of the actual completion of theinternational search | Date of mailing of the international search report |
|---|---|
| 15 September 1998 | 22/09/1998 |

| Name and mailing address of the ISA | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,<br>Fax: (+31-70) 340-3016 | Brandt, J |

Form PCT/ISA/210 (second sheet) (July 1992)

# INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter ..onal Application No
PCT/US 98/12017

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| WO 9704394 A | 06-02-1997 | AU 5945796 A | 23-01-1997 |
| WO 9733216 A | 12-09-1997 | US 5748741 A | 05-05-1998 |

Form PCT/ISA/210 (patent family annex) (July 1992)

フロントページの続き

(81)指定国　　EP(AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OA(BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), AP(GH, GM, KE, LS, MW, SD, SZ, UG, ZW), EA(AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW

(72)発明者　トムボーソン，クラーク　デビット
ニュージーランド国，オークランド，メドウバンクス，ファンコート　ストリート　3／61

(72)発明者　ロウ，ダグラス　ウェイ　コック
ニュージーランド国，オークランド　3，エプソム，アルモラー　ロード　56